TRABAJO DE FIN DE GRADO EN

INGENIERÍA INFORMÁTICA

CURSO 2016-2017

DISEÑO E IMPLEMENTACIÓN DE UNA FUENTE DE DATOS TIPO SNORT BASADA EN LA ARQUITECTURA SELFNET

Daniel Sánchez Álvarez

Gerardo David Reyes Diego

Directores:

Luis Javier García Villalba

Ana Lucila Sandoval Orozco

Departamento de Ingeniería del Software e Inteligencia Artificial

---

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

Abstract

This work is based on the European SELFNET project and is supported by the Horizon 2020 Program of the European Commission under the reference number H2020-ICT-2014-2/671672.

The SELFNET project designs and implements an autonomous network management framework to achieve self-organizing capabilities in the management of network infrastructures. By automatically detecting and mitigating network problems, operators can significantly reduce operational costs.

SELFNET integrates in its architecture innovative technologies such as Software Defined Networks (SDN), Network Function Virtualization (NFV), Self-Organizing Networks (SON), cloud computing, Artificial Intelligence, quality of experience (QoE), to provide a new intelligent framework in network management.

The present work is part of the SELFNET Monitoring architecture. In which, a data source capable of receiving notifications of an SNORT type sensor is designed and implemented. Likewise, the information received is adapted according to a general model of metrics. Finally, metrics are sent to higher layers of aggregation and correlation. The implementation and results demonstrate the efficiency of the present design.

Keywords

Horizon 2020, NFV, QoE, SDN, SELFNET, SNORT, SON.

Resumen

Este trabajo está basado en el proyecto SELFNET y cuenta con el apoyo del Programa Horizonte 2020 de la Comisión Europea con número de referencia H2020-ICT-2014-2/671672.

El proyecto SELFNET diseña e implementa un marco de gestión autónomo de la red para lograr capacidades de auto-organización en la gestión de infraestructuras de red. Mediante la detección y mitigación automática de problemas de red, los operadores pueden reducir significativamente los costes operacionales.

SELFNET integra en su arquitectura tecnologías innovadoras tales como las Redes Definidas por Software (SDN), la Virtualización de Funciones de Red (NFV), las Redes Auto-Organizadas (SON), Computación en la Nube, Inteligencia Artificial, Calidad de Experiencia (QoE), etc., para proporcionar un nuevo marco inteligente en la gestión de redes.

El presente trabajo forma parte de la arquitectura de monitorización de SELFNET, diseñando e implementando una fuente de datos capaz de recibir notificaciones de un sensor tipo SNORT. Asimismo, la información recibida es adaptada según un modelo general de métricas. Finalmente, las métricas son enviadas a capas superiores de agregación y correlación. Los resultados obtenidos demuestran la eficiencia del diseño realizado.

Palabras clave

Horizonte 2020, NFV, QoE, SDN, SELFNET, SNORT, SON.

Agradecimientos

Queremos agradecer especialmente a nuestros directores Luis Javier García Villalba y Ana Lucila Sandoval Orozco por ofrecernos la posibilidad de realizar el presente proyecto de fin de grado.

También queremos dedicar un especial agradecimiento a Marco Antonio Sotelo y Leonardo Valdivieso, pues nos han dedicado incontables horas de su tiempo para desarrollar este proyecto, proporcionándonos información bibliográfica, sus proyectos de investigación y presentaciones propias para adquirir los conocimientos necesarios para afrontar este reto.

Por último, queremos agradecer a nuestros amigos y familiares el apoyo recibido a lo largo del proceso de realización de este proyecto, ya que este supone la culminación de nuestros estudios como ingenieros informáticos.

Daniel Sánchez

Agradezco a mis padres toda la ayuda y compresión que me han dado durante toda mi vida. A Sara por aguantarme los casi dos años que llevamos juntos y que me ha servido de apoyo en momentos difíciles.

Gerardo Reyes

Agradezco de forma muy especial a mi hermano Antonio por haberme apoyado a lo largo de toda mi vida, a mis padres, hermanos, sobrinos y amigos que han hecho más llevadero la vida universitaria gracias a su apoyo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ABREVIATURAS

AAA	*Authentication Authorization Accounting*
API	*Application Programming Interface*
App	*Application*
ARP	*Address Resolution Protocol*
ASCII	*American Standard Code for Information Interchange*
BBU	*Base Band Unit*
BNG	*Broadband Network Gateway*
BRAS	*Broadband Remote Access Server*
BSS	*Business Support System*
CAPEX	*Capital Expenditures*
CDN	*Content Delivery Network*
CG-NAT	*Carrier Grade Network Address Translation*
CPE	*Customer Premises Equipment*
CPU	*Central Processing Unit*
CSV	*Comma-Separated Values*
DDOS	*Distributed Denial Of Service*
DNS	*Domain Name Server*
DPI	*Deep Packet Inspection*
E2E	*End To End*
EM	*Element Management*
EMS	*Element Management System*
EPC	*Evolved Packet Core*

GGSN	*Gateway GPRS Support Node*
GPRS	*General Packet Radio Service*
GUI	*Graphical User Interface*
HLR	*Home Location Register*
HSS	*Home Subscriber Server*
Hw	*Hardware*
IaaS	*Infrastructure as a Service*
IDS	*Intrusion Detection System*
IMS	*Internet Map Service*
IP	*Internet Protocol*
IPS	*Intrusion Prevention System*
IPSec	*Internet Protocol Security*
ISG	*Industry Specification Group*
IT	*Information Technology*
JSON	*JavaScript Object Notation*
KPI	*Key Performance Indicators*
LAN	*Local Area Network*
M2M	*Machine To Machine*
MEC	*Mobile Edge Computing*
MNE	*Mobile Network Emulator*
NaaS	*Network as a Service*
NAT	*Network Address Translation*
NF	*Network Function*
NFaaS	*Network Function as a Service*

NFV	*Network Function Virtualization*
NFV M&O	*Network Function Virtualization Management & Orchestration*
NFVI	*Network Function Virtualization Infrastructure*
NFV-MANO	*Network Function Virtualization Management & Orchestration*
NFVO	*Network Function Virtualization Orchestrator*
NGN	*Next Generation Network*
NOS	*Network Operating System*
NS	*Network Service*
ONF	*Open Network Foundation*
OPEX	*Operating Expenditures*
OSS	*Operations Support System*
PaaS	*Platform as a Service*
PCC	*Policy Control & Charging*
PE	*Provider Edge*
QoE	*Quality of Experience*
QoS	*Quality of Service*
REST	*Representational State Transfer*
RFC	*Request for Comments*
RNC	*Radio Network Controller*
SBC	*Session Border Controller*
SDN	*Software Defined Networking*
SGSN	*Serving GPRS Support Node*
SLA	*Service Level Agreement*
SON	*Self-Organizing Network*

SSH	*Secure Shell*
SSL	*Secure Socket Layer*
Sw	*Software*
TAL	*Tactical Autonomic Language*
TCP	*Transmission Control Protocol*
UC	*Unified Communications*
UCaaS	*Unified Communications as a Service*
UDP	*User Datagram Protocol*
UTF	*Unicode Transformation Format*
VIM	*Virtual Infrastructure Manager*
VNF	*Virtual Network Functions*
VNFM	*Virtual Network Functions Manager*
VPN	*Virtual Private Network*
WAN	*Wide Area Network*
XML	*Extensible Markup Language*

1. INTRODUCCIÓN

Debido al incremento del tráfico de datos y a las nuevas necesidades de comunicación surge un nuevo paradigma. Dentro de este contexto surgen tecnologías como las Redes Definidas por Software o Software Defined Networking (SDN) y la Virtualización de Funciones de Red o Network Function Virtualization (NFV) que son la base para las futuras redes 5G.

SDN se basa en la separación entre el plano de datos y el plano de control, consiguiendo una arquitectura en tres capas:

1. **Plano de datos**, supone el procesamiento de todo el tráfico que circula por la red a través de conmutadores.

2. **Plano de control**, es el encargado de gestionar el comportamiento de la red a través de los controladores SDN.

3. **Capa de aplicación**, donde se encuentran aplicaciones que permitan automatizar tareas de configuración, desplegar y hacer provisión de nuevos servicios dentro de la red, etc.

La comunicación entre las diferentes capas se lleva a cabo mediante dos interfaces: NorthBound y SouthBound API.

1. **NorthBound API**: es la encargada de comunicar el controlador, con las aplicaciones programadas de alto nivel.

2. **SouthBound API**: es la encargada de comunicar los dispositivos de la red que forman el plano de datos, con el controlador que forma parte del plano de control.

Por otro lado, la incorporación de NFV supone dejar de desarrollar aplicaciones de red bajo un hardware específico, que imposibilita su utilización en otros entornos red.

La arquitectura de NFV se compone de tres módulos principales: Infraestructura de NFV (NFVI), VNF y Administrador y Orquestador de NFV (NFV M&O).

1. **NFVI**: este módulo contiene los componentes hardware y software que componen el entorno en el cual se despliegan las Funciones de Red Virtual o Virtual Network Function (VNF).

2. **VNF**: es la implementación de software de una función de red que es capaz de funcionar sobre el NFVI.

3. **NFV M&O**: su tarea es la de orquestar y gestionar del ciclo de vida de los recursos físicos y/o de software, que soportan la virtualización de la infraestructura y la gestión del ciclo de vida de los VNF.

Ambas tecnologías se integran en el proyecto SELFNET, que diseña e implementa una arquitectura de gestión autónoma de la infraestructura de red. Y cuya arquitectura se basa en seis capas:

1. **Capa de Infraestructura**: se basa en los diseños y los componentes que se están utilizando actualmente en las redes 5G.

2. **Capa de Datos de Red**: se instancian las funciones de red y máquinas virtuales con el fin de proporcionar funcionalidades a los usuarios.

3. **Capa de Control SON**: incluye los elementos para la recolección de los datos desde los distintos sensores y las funciones que ejecutan acciones en la red.

4. **Capa Autónoma SON**: encargada de proporcionar la automatización de

las funciones de red.

5. **Capa de Acceso SON**: implementa una interfaz para administrar los servicios expuestos en SELFNET.

6. **Capa de Gestión y Orquestación**: gestiona el control de las infraestructuras de red disponibles en la arquitectura.

Por último, se definirá el diseño y la implementación de una fuente de datos SNORT. Este supone una fuente de datos capaz de recibir notificaciones de un sensor tipo SNORT, del que posteriormente se realizará una tarea de tratamiento de datos con el fin de obtener un fichero de salida que será utilizada para obtener métricas que se utilicen en capas superiores de agregación y correlación.

1.1. Objeto de la Investigación

Para poder garantizar el correcto funcionamiento de la red, será necesario recibir y tratar la información recogida por los sensores de la red.

La finalidad de este proyecto implica recibir la información de estos sensores a través de un bus de datos. A continuación, se filtrará la información necesaria y se añadirán datos provenientes de un fichero de configuración. Por último, el fichero resultante se enviará por un bus exactamente igual que el anterior, para que las capas superiores obtengan las métricas normalizadas.

1.2. Trabajos Relacionados

Existe una gran variedad de aplicaciones que integran tecnologías como SDN y NFV. En la Tabla 1.1 se describen algunos ejemplos.

Aplicación	SDN	NFV	Descripción
DEMon [AP1]	Si	No	Utilizada para monitorizar el centro de datos de Microsoft.
F5´s Big IP [AP1]	Si	No	Es una aplicación que monitoriza diferentes tipos de amenazas en la red.
Kemp [AP2]	Si	No	Proporciona un equilibrio de carga adaptativo en función del tráfico en los conmutadores.
Juniper [JUN15]	No	Si	Combina un sistema de gestión de aplicaciones en la nube y una arquitectura de referencia de apoyo.
Masergy [TRU14]	No	Si	Proporciona un análisis de comportamiento de red y una plataforma de seguridad basada en correlación.
Velocloud [VEL]	No	Si	Simplifica la gestión de la red mediante la automatización del despliegue y la mejora del rendimiento.
FastStream Platform [FST]	Si	Si	Incluye cliente OpenFlow, enrutamiento híbrido, framework de conmutación, framework de automatización de pruebas Openflow.
Mellanox [MEL16]	Si	Si	Aumento de rendimiento a través de adaptadores inteligentes, incrementa la escalabilidad y reduce el coste de despliegue.
Brain4Net [B4N16]	Si	Si	Suministra servicios de adaptación a SDN y NFV.

Tabla 1.1: Comparativa de aplicaciones SDN y NFV.

1.3. Estructura del Trabajo

Este proyecto está dividido en ocho capítulos, descritos a continuación.

El Capítulo 2 explica detalladamente el funcionamiento y la arquitectura de las redes SDN.

El Capítulo 3 define el concepto de NFV y los tres módulos principales de su arquitectura.

El Capítulo 4 describe el proyecto SELFNET junto con su arquitectura y sus dos principales tareas que son la monitorización y el descubrimiento de las métricas generadas por la infraestructura virtualizada.

El Capítulo 5 describe el diseño y la implementación de una fuente de datos SNORT.

En el Capítulo 6 se explican los test realizados y los resultados obtenidos tras la implementación de una fuente de datos SNORT.

En el Capítulo 7 se exponen las principales conclusiones y trabajos futuros. De igual manera, se describe un resumen con la información relevante del estudio.

Finalmente, en el Capítulo 8 se muestra la división del trabajo realizada por cada uno de los integrantes del proyecto.

2. Redes Definidas Por Software

2.1. Definición

Las Redes Definidas por Software (del inglés Software Defined Networking) se diferencian de las arquitecturas tradicionales con el fin de adaptarse a los nuevos requerimientos de servicios red.

En SDN, el plano de datos se encuentra separado del plano de control. De este modo se obtiene mayor progamabilidad y flexibilidad. La arquitectura SDN se encuentra descrita en la Figura 2.1.

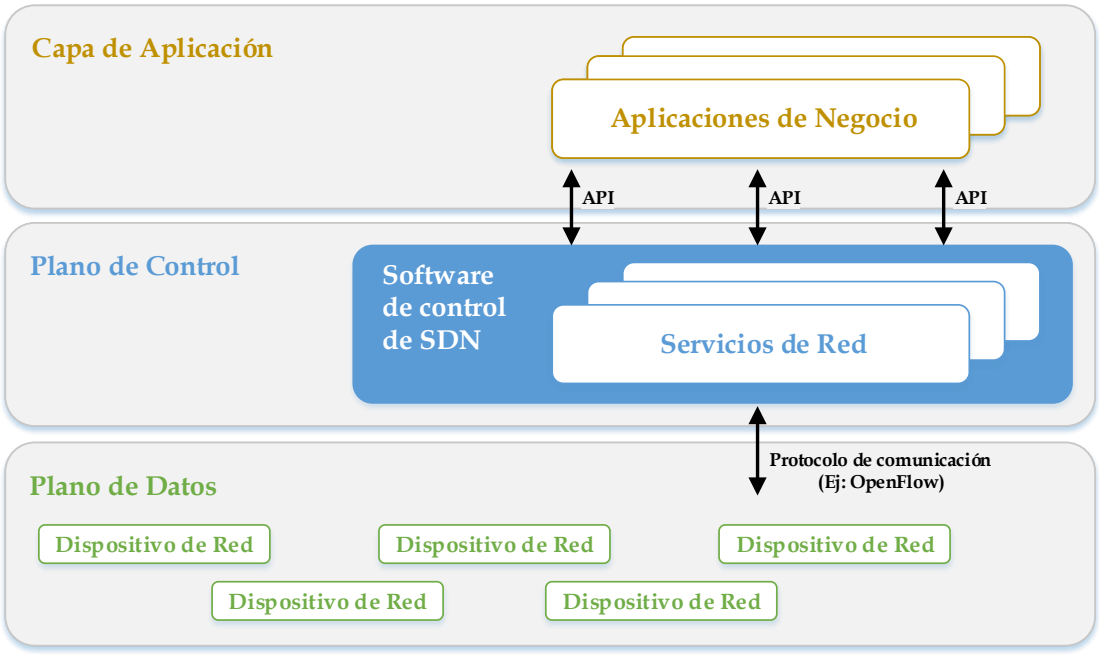

Figura 2.1: Arquitectura SDN. [ONF1]

2.2 Limitaciones de las Arquitecturas Tradicionales

La innovación en los servicios de red se ha visto limitada por la rigidez de las arquitecturas tradicionales. La estrecha unión entre plano de datos y plano de control dificulta la creación de nuevas aplicaciones. En este sentido, a

continuación, se describen las principales limitaciones de las arquitecturas tradicionales.

- **No son flexibles**: se comportan en base a reglas o protocolos que incorporan los fabricantes de dispositivos.
- **Están basadas en multitud de reglas predefinidas**: los protocolos actuales están basados en más de 5000 RFC, y cualquier modificación requiere de procesos complejos cuya duración puede ser de algunos meses o años.
- **Limitaciones en investigación y desarrollo**: la implementación de nuevos servicios por parte de desarrolladores externos puede causar problemas con los servicios que funcionan adecuadamente.
- **Recursos de operación y mantenimiento**: el número de responsables de control y mantenimiento de la red es muy elevado, ya que no existe un proceso automatizado.
- **Demora al actualizar la red**: se invierte mucho tiempo a la hora de introducir o reconfigurar elementos en una red, puesto que se hace de manera manual.
- **Ineficacia al manejar grandes volúmenes de datos**: con el auge actual del Big Data, se requiere una respuesta ágil frente a la demanda de servicios, lo cual necesita un gran ancho de banda para poder transmitir grandes cantidades de información a través de la red.

La Tabla 2.1 muestra las diferencias entre ambos modelos de redes, en las que se puede observar el por qué ha surgido un nuevo paradigma.

	Redes Tradicionales	SDN
Plano de Datos y Plano de Control	Ambos situados en los elementos de la red.	Plano de datos separado del plano de control.
Control	Distribuida por los elementos de la red.	Centralizada en los controladores SDN.
Programación de red y aplicaciones	No se puede programar la red mediante aplicaciones. Cada elemento de la red se configura	La red se puede programar. El controlador puede exponer interfaces de aplicación para

	por separado.	la manipulación de la red.

Tabla 2.1: Diferencias entre redes tradicionales y SDN.

2.3 Arquitectura de la Redes Definidas por Software

Las redes SDN se basan en la separación del plano de datos, del plano de control y las aplicaciones que interactúan con la red.

La ONF (Open Network Foundation) [ONF1] define su arquitectura en tres capas:

1. **Capa de Aplicación:** consiste en las aplicaciones de negocio de los usuarios finales, que utilizan los servicios de comunicación de SDN a través de las API (Northbound) de la capa de control, como JSON, REST, XML, etc.
2. **Plano de Control:** esta capa comprende un conjunto de controladores SDN, cada uno de los cuales tiene control exclusivo sobre un conjunto de recursos expuestos por uno o más elementos de la red.
3. **Plano de Datos:** está compuesta de uno o más dispositivos de red, que son los encargados de redirigir el tráfico que pasa por ellos según las directrices de los controladores.

2.3.1 El Plano de Datos

La función del plano de datos es la de procesar todo el tráfico que llega a los dispositivos de red, como puede ser por ejemplo un conmutador.

Debido a la separación del plano de datos del plano de control, los dispositivos de red ya no son los encargados de decidir el siguiente salto de los paquetes entrantes. En su lugar, el conmutador envía la información relevante al controlador. El controlador decide las acciones a ejecutarse (ej. el puerto de salida). Después, el controlador indicará al dispositivo que acciones deberá

tomar para que finalmente el conmutador las aplique, como se puede observar en la Figura 2.2. Existe una gran variedad de acciones a realizar por parte del controlador (reenviar el paquete a través de un puerto, encapsular y reenviar el paquete al controlador, eliminar el paquete, enviarlo a la pila de procesamiento normal, etc), por lo que le dotará de gran versatilidad y programabilidad.

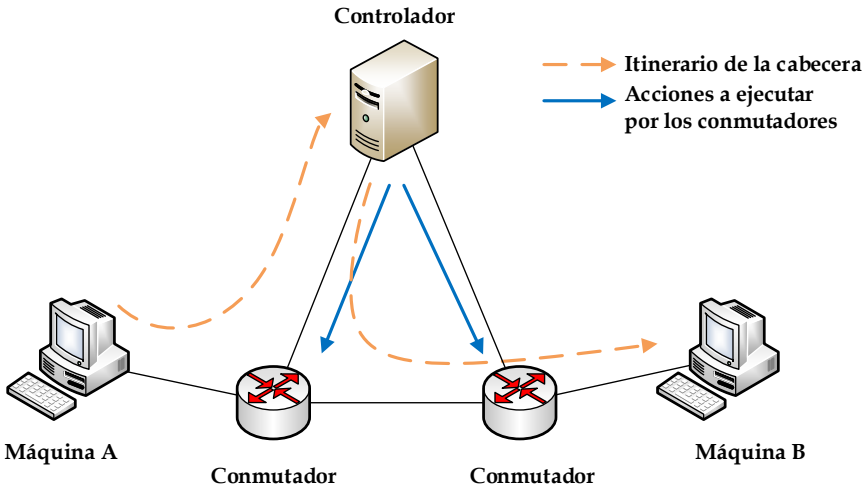

Figura 2.2: Plano de Datos.

2.3.2 El Plano de Control

El elemento clave en esta capa es el controlador, que es el encargado de configurar cada dispositivo existente en la red y de programar el reenvío de flujos de manera automática, teniendo en cuenta factores como la congestión de la red. La principal ventaja de las redes SDN en comparación con las redes tradicionales, es que bastaría con programar adecuadamente el controlador para que el funcionamiento de la red fuera el idóneo.

Al controlador se le denomina Sistema Operativo de Red (del inglés Network Operating System NOS), y este puede ser tanto distribuido como centralizado. Las redes SDN ofrecen múltiples ventajas (escalabilidad, flexibilidad automatización). Sin embargo, los fallos en el controlador pueden comprometer el óptimo funcionamiento de toda la red. En la Figura 2.3 se describen algunas

de las amenazas más comunes en las redes SDN. Al ser un elemento que concentra todo el control en un único elemento, éste debe ser asegurado por todos los medios posibles para no incurrir en un fallo que causaría el mal funcionamiento de toda la red.

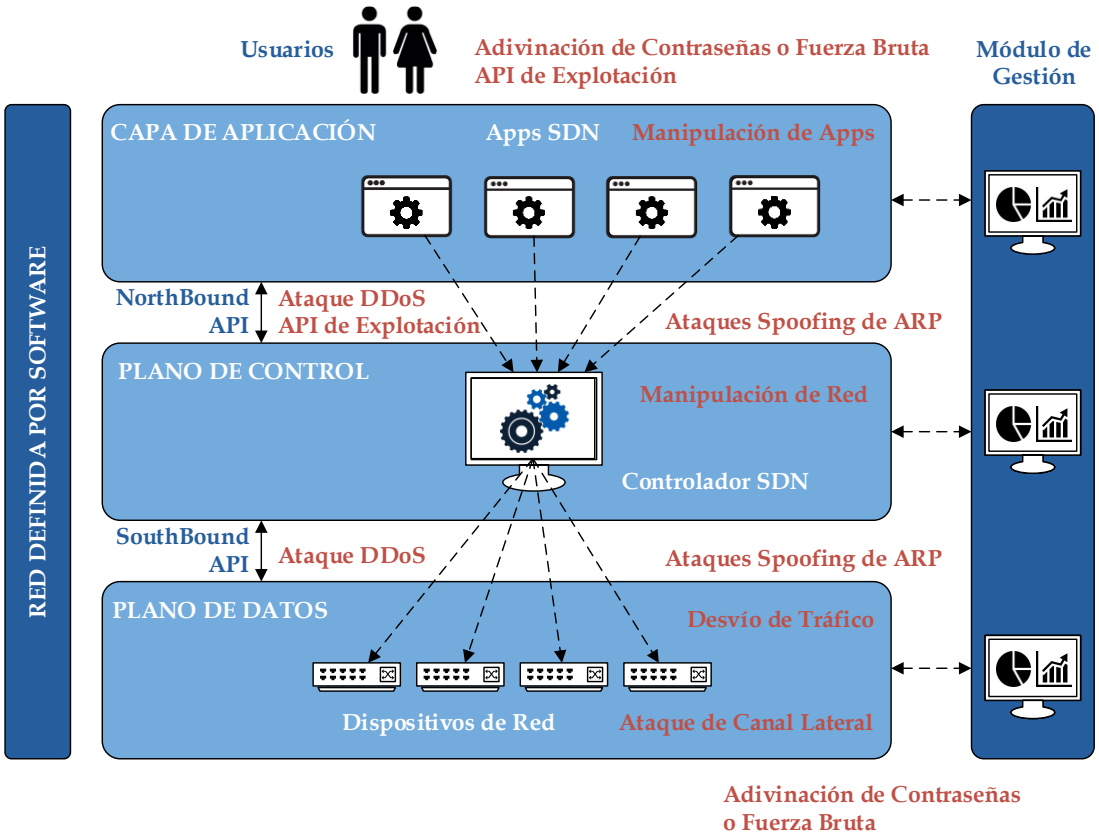

Figura 2.3: Posibles ataques a una red SDN. [ATK]

El controlador es la pieza central dentro de las redes SDN, y por tanto necesita de dos interfaces para la comunicación tanto con el plano de datos como con la capa de aplicaciones. Estas dos interfaces reciben el nombre de Northbound y Southbound, como se puede observar en la Figura 2.4.

La interfaz Northbound es la que permite el desarrollo de aplicaciones de alto nivel. Como por ejemplo NaaS, la integración de sistemas de seguridad, servicios computacionales, servicios de monitorización de redes, etc.

En el otro lado se encuentra la interfaz Southbound que se encarga de

comunicar el plano de datos con el plano de control. Esta interfaz facilita que exista un control eficiente sobre la red, y consiga que el controlador pueda hacer las modificaciones pertinentes de acuerdo a la demanda de necesidades en tiempo real. La interfaz Southbound que tiene una mayor relevancia en la actualidad es OpenFlow, la cual se desarrollará más adelante.

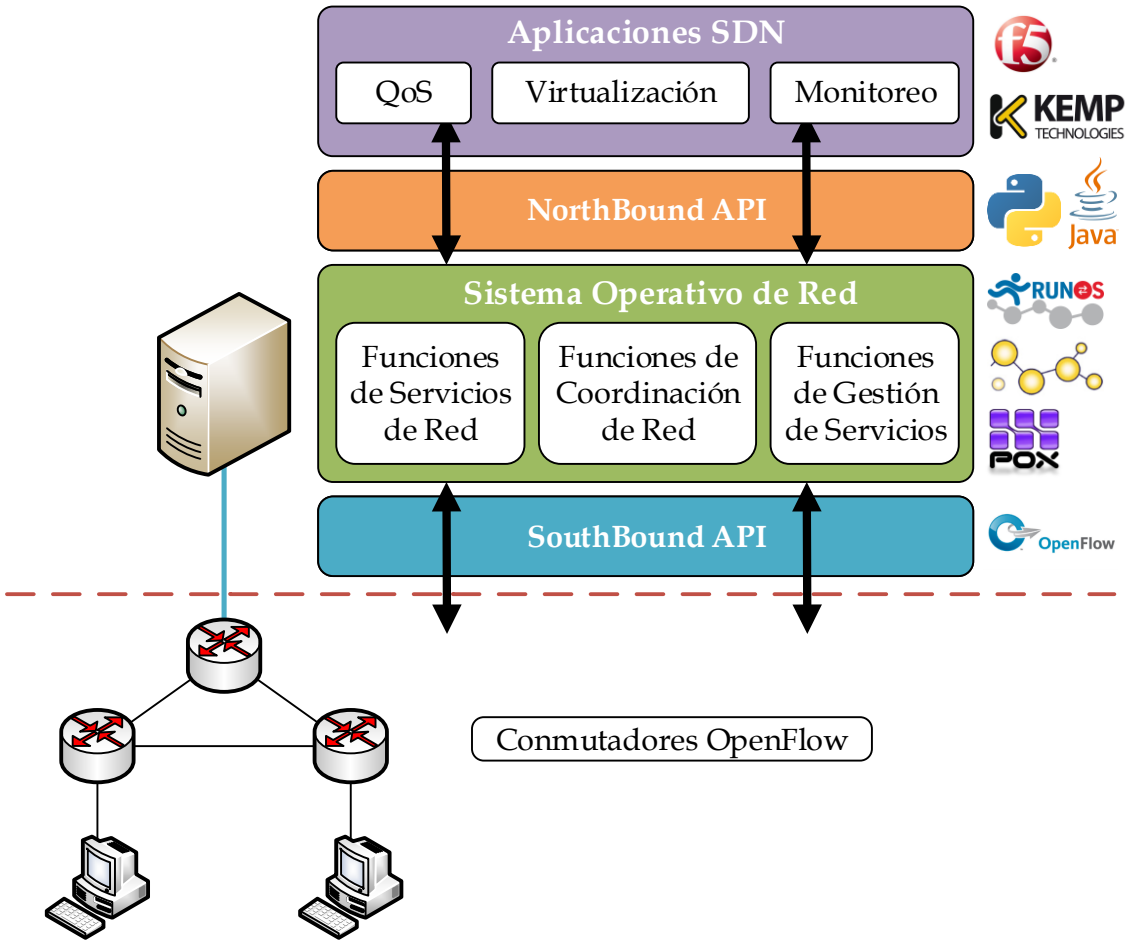

Figura 2.4: Interfaces del controlador SDN.

Los controladores se pueden clasificar según el lenguaje de programación utilizado, tal como se resume en la Tabla 2.2.

Lenguaje de programación	Controlador
C	MUL
C++	NOX, LibFluid, Runos
Python	POX, RYU
Java	OpenDayLight, Beacon, FloodLight, Iris, ONOS

Tabla 2.2: Controladores según el lenguaje de programación.

2.3.3 Capa de Aplicación

La presente capa permite crear aplicaciones para automatizar tareas de configuración, desplegar y hacer provisión de nuevos servicios dentro de la red, etc. La mayor parte de aplicaciones destinadas a funcionar en las redes SDN, se pueden agrupar principalmente en: aplicaciones de monitorización, ingeniería de tráfico y virtualización de redes, entre otros.

A continuación se muestran algunos ejemplos de estas aplicaciones [AP1,AP2]:

1. *DEMon*: Microsoft utiliza el sistema DEMon(Distributed Ethernet Monitoring) para monitorizar su centro de datos.
2. *F5´s Big IP*: es una aplicación DDoS que monitoriza diferentes tipos amenazas.
3. *Kemp*: proporciona un equilibrio de carga adaptativo basado en la carga de trafico de los conmutadores.
4. *Ecode Evolve*: está orientada al diseño de redes, monitorización, solución de problemas, simulación y optimización de rendimiento.

2.4 Protocolo OpenFlow

El protocolo OpenFlow [ONF2] está considerado como uno de los primeros estándares SDN. Openflow permite interactuar directamente con los dispositivos de red, tanto físicos como virtuales, para que pudiera adaptarse mejor a las necesidades cambiantes del entorno.

A través de esta interfaz, el controlador simplifica los cambios a realizar en las tablas de flujo del conmutador, permitiendo a los administradores de red dividir el tráfico, controlar los flujos para un rendimiento óptimo y comenzar a probar nuevas configuraciones y aplicaciones. Los principales mensajes del protocolo Openflow son: asíncronos, simétricos y de controlador a conmutador.

2.4.1 Conmutador OpenFlow

Un conmutador OpenFlow está compuesto por dos elementos: una tabla de flujo (*flow table*) y un canal seguro de comunicación (*secure channel*) por el que se conecta al controlador, como se describe en la Figura 2.5.

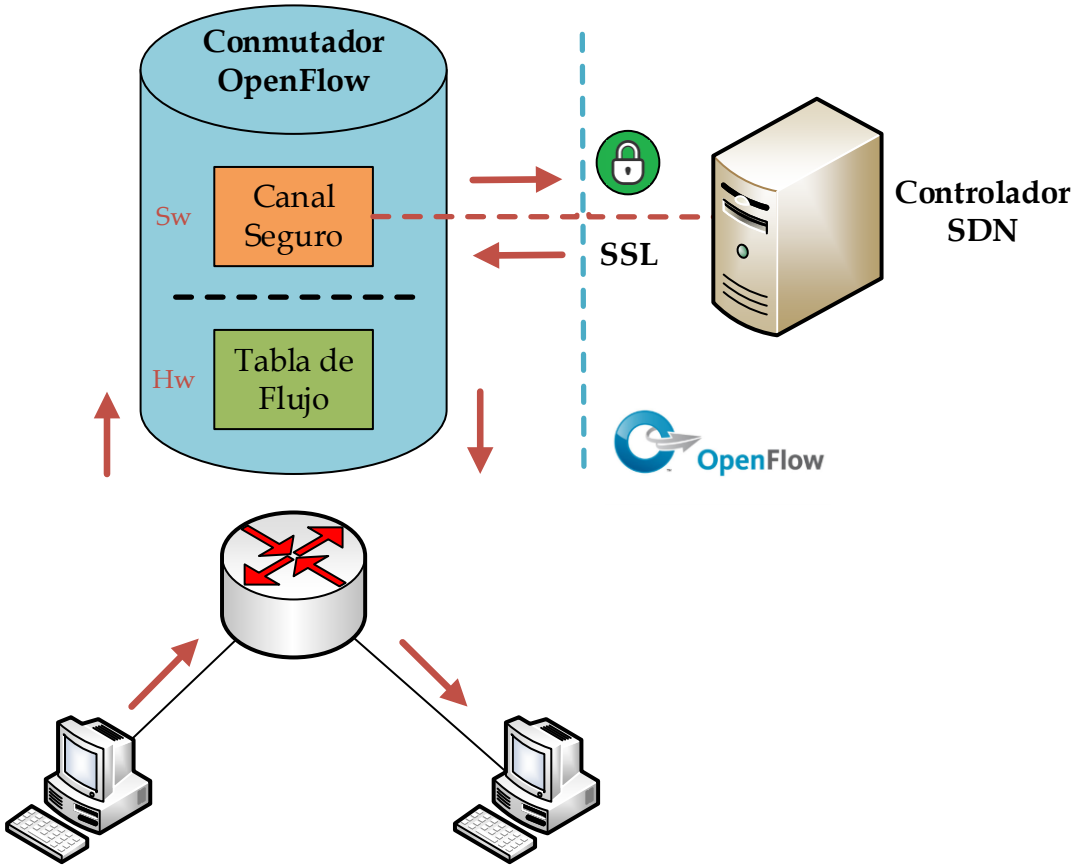

Figura 2.5: Conmutador OpenFlow.

2.4.2 Tablas de Flujo

Una tabla de flujo de un conmutador OpenFlow, como se puede observar en la Figura 2.6 tiene la siguiente estructura:

- *Campos de cabecera*: se utiliza para encontrar una coincidencia entre paquetes.
- *Contadores*: se utilizan para llevar el registro de los paquetes han coincidido.

- *Acciones*: son el conjunto de acciones que se realizaran en el momento de encontrar una coincidencia entre paquetes.

Figura 2.6: Estructura de la Tabla de Flujo.

Cada entrada de flujo contiene un conjunto de instrucciones que se ejecutan cuando un paquete coincide con la entrada. Estas instrucciones dan lugar a cambios en el paquete, un conjunto de acciones y al procesamiento del pipeline.

2.4.2.1 Acciones

El paquete entrante es comparado con las entradas de una tabla de flujo y en caso de encontrar una coincidencia, se realizarán las acciones pertinentes. Por el contrario, si no existe ninguna coincidencia con alguna de las entradas de la tabla de flujo, el conmutador o bien descartará el paquete, o bien lo encapsulará y lo enviará al controlador, en función de su configuración previa.

El conmutador OpenFlow no necesita soportar todos los tipos de acciones, sólo las que estén marcadas como "*Required Action*". Adicionalmente el controlador podrá consultar al conmutador sobre cuáles de las "*Optional Action*" soporta. Las acciones más utilizadas son: forward, drop y queue.

2.4.3 Mensajes OpenFlow

El canal de comunicación entre el controlador y los conmutadores OpenFlow, soporta una gran variedad de mensajes, y se pueden dividir en:

- Mensajes de **controlador a conmutador:** son iniciados por el controlador y puede que no necesiten respuesta del conmutador. Entre los cuales destacan: *Packet-Out*, *Features* y *Configuration*.

- Mensajes **asíncronos**: son enviados sin que el controlador los solicite desde un conmutador, como *Packet-In* y *Flow-Mod*.
- Mensajes **simétricos**: se mandan sin necesidad de enviar una solicitud, e independientemente de si se envían del controlador al conmutador o viceversa. Se pueden destacar los mensajes *Hello*, *Echo* y *Error*.

2.5 Herramientas

Para la implementación de una red SDN se necesita al menos un controlador, y la elección de este se hará en función de las necesidades del usuario, así como un entorno de red virtualizado para hacer las pruebas pertinentes.

2.5.1 Controladores

El controlador es el elemento clave que comunica el plano de datos con el plano de control. Este permite desarrollar las aplicaciones que se van a ejecutar dentro de la red SDN. Por lo tanto, se puede decir que el controlador es un conjunto de sistemas que otorgan:

- Un canal seguro entre el controlador y los dispositivos de la red.
- Realizan diversas tareas de red como, cuantificar la cantidad de dispositivos que hay en la red, recopilar información acerca de las funcionalidades de los dispositivos, generar estadísticas, etc.
- Disponer de un API REST, que supone una capa de abstracción para que las aplicaciones puedan hacer uso de las funcionalidades del controlador.

Los controladores más utilizados en la actualidad se describen en la Tabla 2.3.

	ODL	FloodLight	ONOS	POX	NOX	MUL	Ryu
Lenguaje de Programación	Java	Java	Java	Python	C++	C	Python
GUI	Basado en Web	Basado en Web	Basado en Web	Python + QT4	Python + QT4	Basado en Web	Si
Documentación	Muy Buena	Buena	Buena	Mala	Mala	Media	Media
Modularidad	Alta	Media	Alta	Baja	Baja	Media	Media
Distribuida o Centralizada	D	C	D	C	C	C	C
Soporte de Plataforma	Linux, MAC y Windows	Linux, MAC y Windows	Linux, MAC y Windows	Linux, MAC y Windows	Soporte mayoritario en Linux	Soporte mayoritario en Linux	Soporte mayoritario en Linux
Productividad	Media	Media	Media	Alta	Media	Media	Media
SouthBound API	OF 1.0, 1.3, 1.4	OF 1.0, 1.3	OF 1.0, 1.3	OF 1.0	OF 1.0	OF 1.0, 1.3, 1.4	OF 1.0, 1.2, 1.3, 1.4
NorthBound API	REST API	REST API	REST API	REST API	REST API	REST API	REST API
Soporta Multihilo	Si	Si	Si	No	NOX_MT	Si	Si
Soporta OpenStack	Si	No	No	No	No	Si	Si

Tabla 2.3: Características de los Diferentes Controladores SDN. [RSG]

2.5.2 Mininet [MIN]

Mininet es un emulador de red que ejecuta una colección de máquinas, conmutadores, enrutadores y enlaces en un único kernel de Linux. Se pueden utilizar conexiones SSH en él, y ejecutar programas arbitrarios. Los programas que se ejecutan pueden mandar paquetes, con una velocidad de enlace y retraso asignado previamente. Los paquetes son procesados como si de una red real se tratase, con sus consiguientes colas a la hora de procesar los paquetes. En la Figura 2.7 se representa una red con dos host (h1, h2) y un switch (s1) con dos interfaces (s1-eth1, s1-eth2).

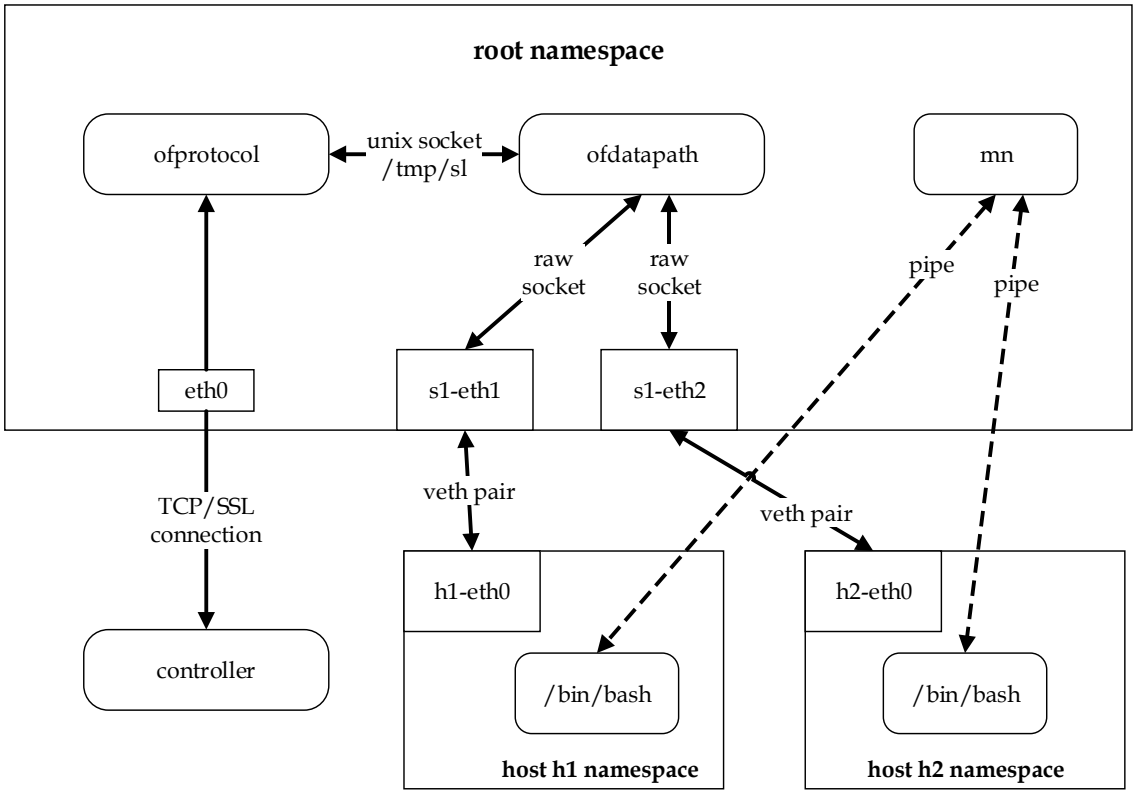

Figura 2.7: Red Mininet formada por dos máquinas.

Se pueden crear topologías lineales, simples, en forma de árbol, o bien se pueden programar de una forma específica según las necesidades.

2.5.2.1 Ventajas

Mininet presenta multitud de ventajas a la hora de emular una red, y son las siguientes:

- Es **rápido**: iniciar una red sencilla tarda unos segundos.
- Puedes **crear topologías personalizadas**: un solo conmutador, grandes topologías como Internet, un centro de datos, etc.
- Puedes **ejecutar programas reales**: cualquier cosa que funcione en Linux está disponible para ejecutarlo, desde servidores web hasta herramientas TCP de monitoreo como Wireshark.
- Puedes **personalizar el reenvío de paquetes**: los conmutadores Mininet se programan usando el protocolo OpenFlow, por lo que se pueden transferir los diseños de red fácilmente a conmutadores OpenFlow.
- Los **resultados se pueden compartir y replicar**: cualquiera con una computadora puede ejecutar un código una vez empaquetado.
- Se puede **usar con facilidad**: se puede crear y ejecutar experimentos en Mininet simplemente escribiendo un script Python.

2.5.2.2 Limitaciones

Las ventajas de emular redes con Mininet son muy amplias, sin embargo, existen algunas limitaciones:

- Ejecutar Mininet en un solo sistema es conveniente, pero impone una limitación de los recursos.
- Mininet utiliza un único kernel de Linux para todas las máquinas virtuales, por lo que no se pueden ejecutar programas que dependan un kernel de Windows o cualquier otro.
- Mininet no puede escribir un controlador OpenFlow por sí mismo, por tanto, si la red es personalizada, se deberá desarrollar mediante código.

- Por defecto, la red Mininet está aislada de la LAN y de Internet. En principio esto es bueno, pero si se quiere usar NAT para conectar la red Mininet con la LAN, se deberá añadir la opción --nat.
- Por defecto, todas las máquinas Mininet comparten el sistema de ficheros y el espacio PID, esto significa que hay que tener cuidado si se están ejecutando demonios que requieren configuración en /etc, además de no matar ningún proceso por error.
- A diferencia de un emulador, Mininet no tiene una gran noción del tiempo virtual, por lo que las mediciones de tiempo se basarán en tiempo real, y por tanto esos resultados en tiempo real no podrán ser emulados con facilidad.

3. VIRTUALIZACIÓN DE FUNCIONES DE RED NFV

3.1 Introducción

La virtualización de redes no es un concepto nuevo, ya que viene usándose durante años sobre todo para crear entornos controlados de pruebas. El término virtualización [RS13] se refiere a la abstracción de los recursos lógicos frente a los recursos físicos, creando múltiples instancias lógicas sobre la misma infraestructura física. En otras palabras, se pueden emular múltiples instancias de infraestructuras de red sobre una única red física.

NFV [GARG14] reduce el coste de despliegue y operativo, y aumenta la capacidad de administración e innovación en el espacio de servicio de las funciones de red. Esta tecnología ofrece una nueva forma de diseñar, implementar y gestionar servicios de red, desacoplando las funciones de red de las aplicaciones de hardware propietarias para que puedan funcionar con software.

En la actualidad, [MSG15, GUJ16] las grandes empresas invierten grandes cantidades de dinero en la creación de nuevas funcionalidades de red o en la actualización de las mismas. Esto implica que no pueden ser usadas por terceros, ya que están implementadas para un hardware específico y que por tanto hace que sean poco flexibles y escalables. En este contexto, NFV propone la transferencia de las funcionalidades de red (enrutamiento, cortafuegos, inspección profunda de paquetes, puertas de enlace) hacia aplicaciones virtuales basadas en software, que son ejecutadas en plataformas IT (servidores, conmutadores y dispositivos de almacenamiento). Esta nueva visión de los servicios de IT ofrecen una mayor flexibilidad, escalabilidad y facilita el desarrollo de nuevas Funciones de Red Virtuales (VNF) a la vez que disminuyen los costes. En la Figura 3.1 se puede ver la diferencia entre las redes tradicionales y NFV.

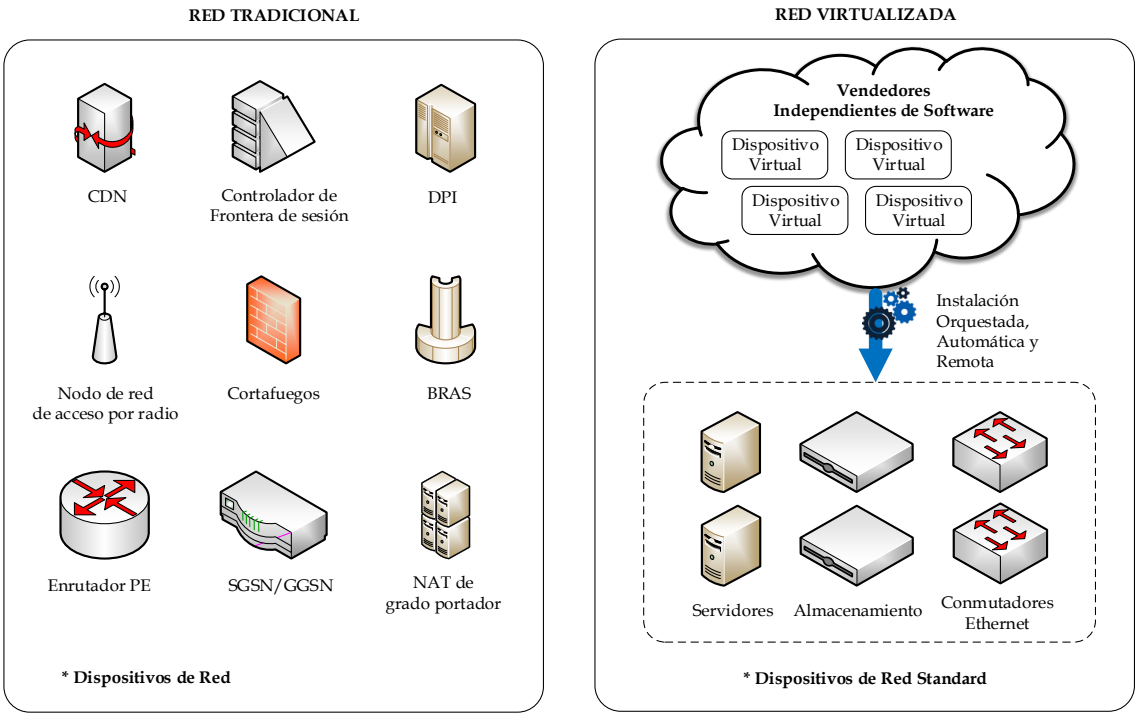

Figura 3.1: Red Tradicional vs Red Virtualizada.

3.2 Arquitectura

El Instituto Europeo de Normas de Telecomunicaciones (del inglés European Telecommunication Standards Institute ETSI) [ETSI14] promueve la estandarización de NFV.

La arquitectura de NFV se compone de tres módulos principales: Infraestructura de NFV (NFVI), VNF y Administrador y Orquestador de NFV (NFV M&O), como se puede observar en la Figura 3.2.

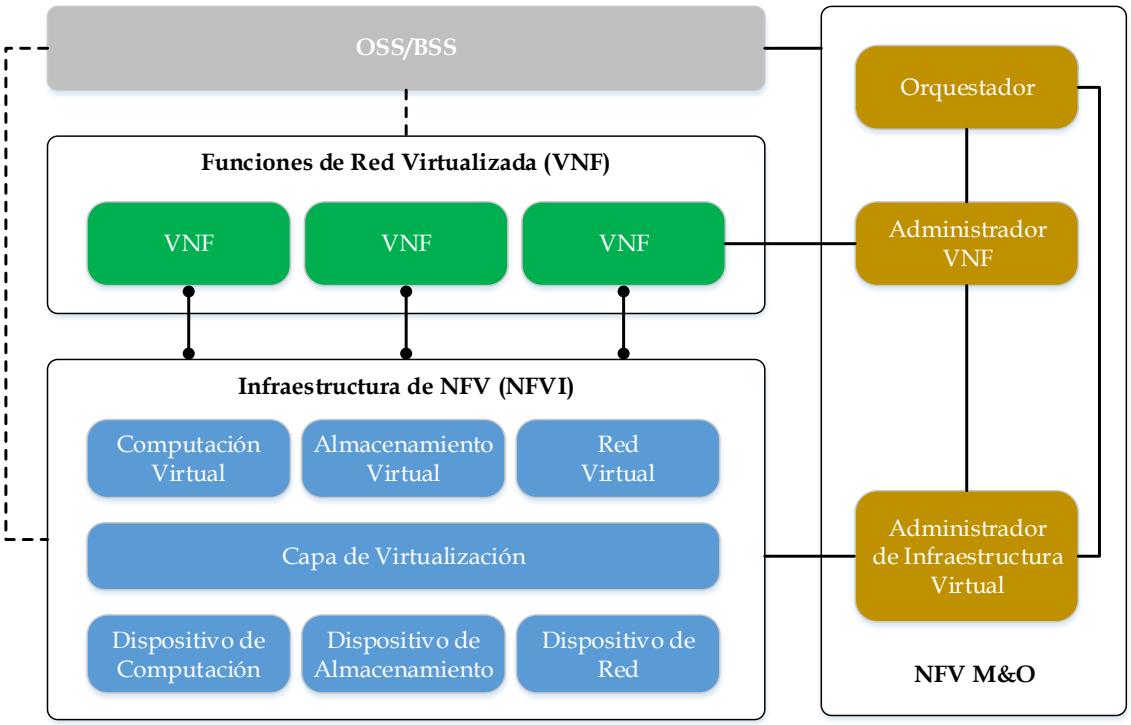

Figura 3.2: Arquitectura NFV. [NFV1]

3.2.1 NFVI

La Infraestructura de NFV [NFV2] es necesaria para soportar la amplia gama de casos de uso y campos de aplicación ya identificados por el NFV ISG, a la vez que proporciona una plataforma estable para la evolución del ecosistema VNF. El NFVI supone la totalidad de los componentes hardware y software que componen el entorno en el cual se despliegan los VNFs. NFVI proporciona una infraestructura multi-usuario aprovechando la tecnología estándar IT de virtualización, que puede soportar múltiples casos de uso y campos de aplicación simultáneamente.

NFVI se implementa como un conjunto de nodos NFVI desplegados en varios puntos de presencia NFVI, según se requiera para soportar los objetivos de localidad y latencia de los diferentes casos de uso y campos de aplicación. Los VNFs pueden ser desplegados dinámicamente en el NFVI bajo demanda dentro los límites de capacidad de los nodos NFVI.

3.2.2 NFV M&O

También conocido como NFV-MANO [NFV1], cubre la orquestación y la gestión del ciclo de vida de los recursos físicos y/o de software, que soportan la virtualización de la infraestructura y la gestión del ciclo de vida de los VNF. El Administrador y Orquestador de NFV se centra en las tareas de gestión específicas de la virtualización necesarias en el marco del NFV. El NFV M&O también interactúa con el (NFV externo) OSS/BSS, lo cual permite que NFV pueda ser integrado en un panorama de gestión ya existente en toda la red.

El marco arquitectónico NFV M&O [NFV3] se centra en los aspectos introducidos en las redes del operador por NFV, por tanto principalmente en los nuevos bloques funcionales y los puntos de referencia entre esos bloques.

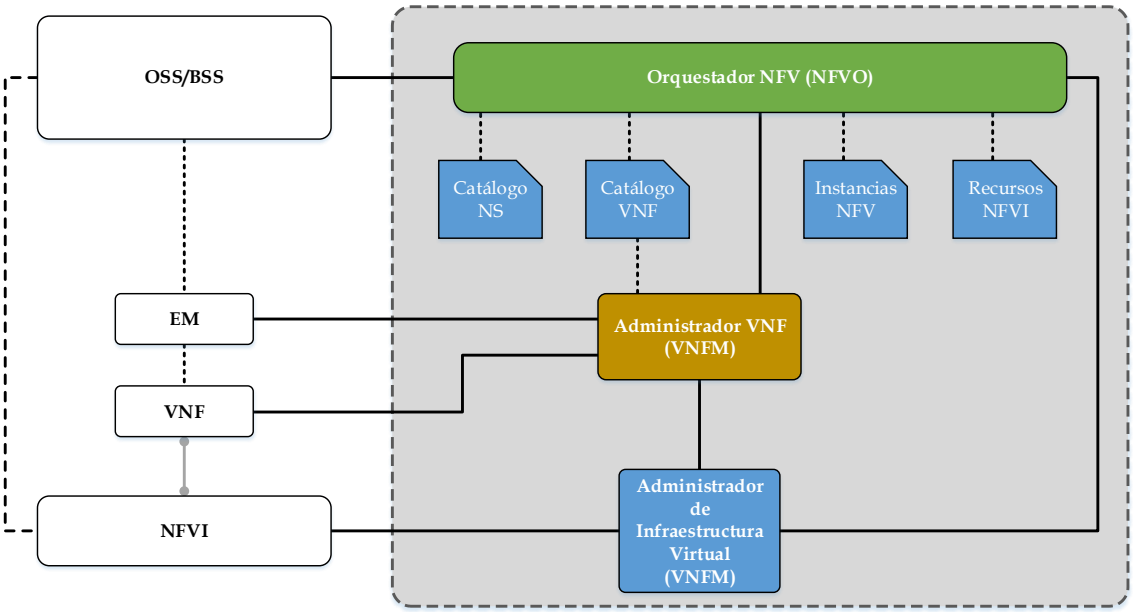

Figura 3.3: Arquitectura del Administrador y Orquestador de NFV. [NFV3]

En el NFV-MANO se pueden identificar tres bloques principalmente, como se puede observar en la Figura 3.3:

- Orquestador NFV (NFVO).
- Administrador VNF (VNFM).
- Administrador de Infraestructura Virtual (VIM).

3.2.2.1 NFVO

El orquestador de NFV tiene las siguientes responsabilidades [NFV3]:

- La orquestación de recursos NFVI a través de múltiples VIMs, cumpliendo con las funciones de orquestación de recursos.
- La gestión del ciclo de vida de los servicios de red, cumpliendo con las funciones de orquestación de servicios de red.

3.2.2.2 VNFM

El Administrador de VNF (VNFM) [NFV3] es responsable de la gestión del ciclo de vida de las instancias de VNF. Se supone que cada instancia tiene un VNFM asociado. A un VNFM se le puede asignar la gestión de una única instancia VNF o la gestión de múltiples instancias VNF del mismo tipo o de tipos diferentes.

Los VNFM son funciones comunes genéricas aplicables a cualquier tipo de VNF. Sin embargo el marco arquitectónico del VNF-MANO también debe admitir casos en los que las instancias VNF necesitan funciones específicas para su gestión del ciclo de vida, y tal funcionalidad puede especificarse en el paquete VNF.

3.2.2.3 VIM

El Administrador de Infraestructura Virtual (VIM) [NFV3] es responsable de controlar y administrar los recursos de computación, almacenamiento y red NFVI, normalmente dentro del Dominio de Infraestructura de un operador. Un VIM puede estar especializado en el manejo de cierto tipo de recursos NFVI, o puede ser capaz de administrar múltiples tipos de recursos NFVI. Por ejemplo, un VIM expone interfaces NorthBound que soportan la gestión de recursos virtualizados de computación, almacenamiento y recursos de red.

Las interfaces SouthBound interactúan con una gran variedad de hipervisores y Controladores de Red con el fin de realizar la funcionalidad expuesta a través de sus interfaces NorthBound. Otras implementaciones de VIM pueden exponer directamente las interfaces expuestas por dichos controladores de red, computación o almacenamiento como VIMs especializados.

3.2.3 VNF

La Función de red Virtualizada (VNF) [NFV3] es la implementación de software de una función de red que es capaz de funcionar sobre el NFVI. Una NFV puede ser acompañada por un Sistema de Gestión de Elementos (EMS), siempre y cuando sea aplicable a la función particular, que entiende y gestiona a un VNF individual y sus peculiaridades. El VNF es la entidad que corresponde a los nodos de red actuales, que ahora se espera que se entreguen como software libre de dependencias de hardware.

3.3 Relación con SDN [NFV2]

Como se puede observar en la Figura 3.4, NFV se complementa a la perfección con SDN. Cabe destacar que no existe una relación de dependencia entre ambas, ya que pueden funcionar correctamente de manera aislada.

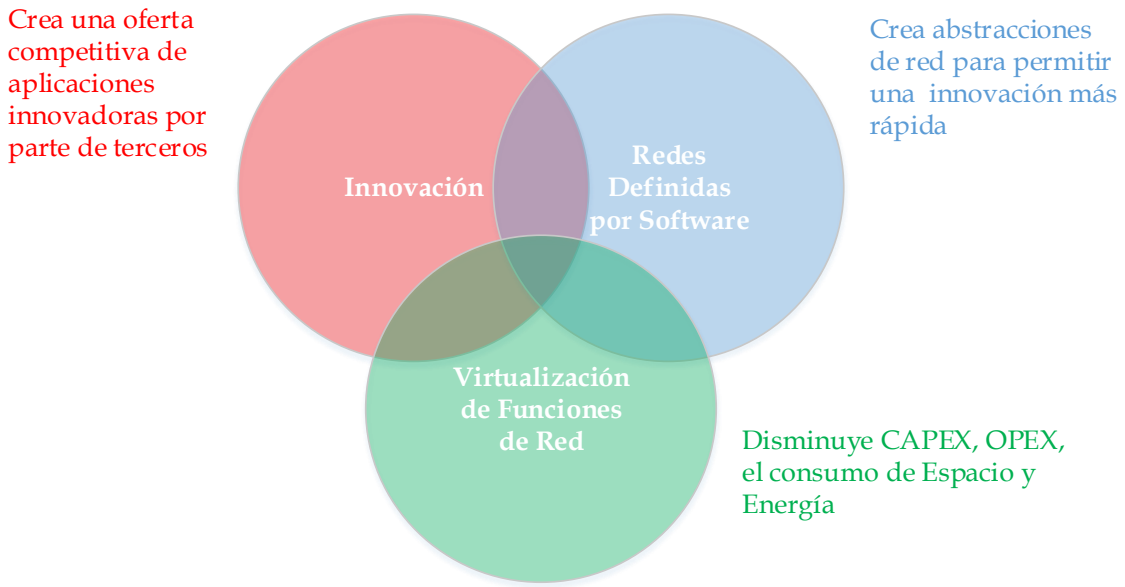

Figura 3.4 Relación de NFV con SDN. [NFV4]

Los dos elementos clave [NFV4] de SDN son la separación del plano de datos del plano de control, para formar una vista de todo el dominio de una red y la capacidad de abstraer y controlar de forma automatizada los recursos de red. Ambas capacidades encajan a la perfección en el paradigma NFV y como tal, SDN puede desempeñar un papel significativo en la orquestación de los recursos de NFV (tanto físicos como virtuales). Esto permite funcionalidades como: aprovisionamiento y configuración de conectividad de red y ancho de banda, automatización de operaciones, seguridad y control de políticas.

NFV crea un entorno de red muy dinámico, impulsado por clientes que necesitan servicios bajo demanda y operadores que necesitan gestionar la utilización y el rendimiento de los servicios. Las redes de usuarios se modifican continuamente en función de las necesidades, por lo que será necesario equilibrar las cargas a través de la infraestructura.

Las complejas topologías de red pueden ser fácilmente construidas para soportar el aprovisionamiento automatizado de multitud de servicios, al tiempo que garantizan una implementación robusta y consistente a nivel de seguridad y otras políticas.

El controlador SDN se corresponde con el concepto general de controlador de red identificado en el marco arquitectónico de NFV como un componente del dominio de la red NFVI. Por tanto, un controlador SDN puede trabajar eficientemente con sistemas de orquestación y controlar tanto los conmutadores físicos como los virtuales, así como proporcionar la supervisión necesaria de toda la red.

SDN también puede beneficiarse de los conceptos introducidos por NFV, tales como: administradores de infraestructura virtualizados, el orquestador que hace que un controlador SDN pueda funcionar en una máquina virtual. Por tanto SDN podría formar parte de un conglomerado de servicios junto con otros VNFs y convirtiéndose en una función de red virtualizada. Así el propio controlador podría ser implementado como un VNF para beneficiarse de características como la fiabilidad y la elasticidad innatas a NFV.

En un futuro, tanto SDN como NFV se volverán indistinguibles entre sí, ya que todo hace pensar que surgirá un nuevo paradigma que unifique las funcionalidades de ambas tecnologías.

3.4 Ventajas

La aplicación de Funciones Virtualizadas de Red aportan un gran número de ventajas a los operadores de red, contribuyendo a un cambio de gran calado en la industria de las telecomunicaciones. Las ventajas [NFV4] de esta tecnología son las siguientes:

- Reducción de los costes de infraestructura y el consumo de energía, gracias a la disposición de la maquinaria necesaria por parte de las grandes compañías de la industria de las telecomunicaciones.

- Aumento de la velocidad de comercialización, debido a las innovaciones llevadas a cabo por los operadores de red.

- No es necesario hacer grandes inversiones basadas en el hardware que posteriormente se aplicaban a un determinado software, lo que implica reducir el ciclo de maduración de las aplicaciones.

- La posibilidad de realizar instalaciones de producción y pruebas en la misma infraestructura hace que la integración sea mucho más eficiente, ya que se reducen los costes de desarrollo y el tiempo de comercialización.

- Un aumento de la escalabilidad, ya que se pueden introducir servicios orientados a una zona geográfica o a un tipo de cliente en concreto. Además, en función de las necesidades de los clientes, se podrán incrementar o disminuir los recursos asociados de manera remota y automática sin necesidad de hacer cambios físicos en el hardware.

- Supone la apertura al mercado de dispositivos virtuales, para favorecer la entrada de desarrolladores de software sin experiencia y al mundo académico. Al fomentar la apertura se consigue ofrecer nuevos y variados servicios y conseguir más ingresos con un menor coste.

- Optimizar de manera automática la configuración del hardware y la topología de la red en tiempo real, en función de patrones reales de tráfico, movilidad y demanda de servicio.

- Apoya el "multi-tenancy" (cuando se ubican varios clientes dentro de una misma infraestructura de un software) permitiendo a los operadores de red proporcionar servicios adaptados y conectividad para múltiples usuarios, aplicaciones o sistemas internos y operadores de red. Gracias a esto, se consigue que todos coexistan en el mismo hardware separados en diferentes dominios, con el consiguiente ahorro de recursos.

- Reducción del consumo de energía mediante la explotación de las funciones de administración de energía en servidores estándar y almacenamiento, así como la consolidación de la carga de trabajo y la optimización de la ubicación. Por ejemplo, se podría concentrar la carga de trabajo en un menor número de servidores durante las horas de menor actividad, así se conseguiría que el resto de servidores se pudieran apagar o mantenerse en modo ahorro de energía.

- Reducción de la variedad de equipos para la planificación y el aprovisionamiento.

- Posibilidad de reparar fallos mediante la reconfiguración automatizada, y trasladar las cargas de trabajo de la red a otros dispositivos mediante la utilización mecanismo de orquestación. Con esto se consigue reducir el coste de las operaciones en cualquier momento del día y cualquier día del año mitigando los fallos automáticamente.

3.5 Limitaciones

Pese a la gran cantidad de ventajas que supone NFV, cabe destacar algunos inconvenientes debido a la escasa madurez de esta tecnología. Se pueden destacar las siguientes limitaciones [NFV4]:

- En la actualidad la venta de dispositivos que ofrecen una funcionalidad basada en una combinación de hardware y software es todavía muy elevada, esto restringe la flexibilidad de los recursos y las hace ineficientes para entornos NFV.

- El reto es definir una interfaz unificada que desacople las instancias de software del hardware. Hay que tener en cuenta que los vendedores de dispositivos y los proveedores de centros de datos, se sitúan en planos diferentes y dependen los unos de los otros.

- NFV está basada en la industria estándar del hardware, por lo que se evita cualquier mejora que implique la utilización de hardware propietario. Esto implica que hay un descenso en el rendimiento del propio hardware y que hay que tener en consideración. El reto estaría en mantener un rendimiento óptimo utilizando de manera apropiada los hipervisores y el software disponible.

- NFV implica una orquestación de los elementos dinámicos de la red, lo que requiere de un modelo de gestión de red que integre tanto los recursos heredados como los virtuales.

- La automatización es el símbolo de la tecnología NFV, y por tanto se requiere que todas las funciones se puedan automatizar. En el momento que algunas de estas funciones no se puedan automatizar repercutirá muy negativamente en términos de escalabilidad.

- Al existir elementos que unifican el control de todos o gran parte de los elementos de la red, implica que pueden ser susceptibles a fallos que generarían un desplome de la red y sus servicios. También hay que tener en cuenta que estos elementos unificadores, pueden ser objeto de ataques que dañen gravemente la integridad de la red.

3.6 Campos de Aplicación y Casos de Uso

NFV es aplicable a cualquier procesamiento de paquetes dispuesto en un plano de datos, soportado por un conjunto de funciones que supone la capa de control y dentro de un ámbito tanto para redes móviles como fijas. Existen muchos campos de aplicación [NFV4] como por ejemplo:

- Elementos de conmutación como BNG, CG-NAT, encaminadores.

- Nodos de red móvil: HLR/HSS, MNE, SGSN, RNC.

- Análisis de tráfico: DPI, medición de QoE.
- Garantía de servicio, monitorización de SLA, prueba y diagnóstico.
- Señalización de NGN: SBC, IMS.
- Funciones convergente y de red: servidores AAA, control de políticas y plataformas de carga.
- Optimización a nivel de aplicación: CDN servidores de caché, equilibradores de carga, aceleradores de aplicaciones.
- Funciones de seguridad: cortafuegos, escáneres de virus, IDS, protectores de spam.
- Elementos de puerta de enlace para realizar túneles: IPSec/SSL puertas de enlace VPN.
- Funciones contenidas en encaminadores domésticos para crear entornos domésticos virtualizados.

Algunos de los casos de uso [NFV4] que implican mayores beneficios podrían ser los siguientes:

- Un DPI basado en software, que proporciona un análisis de tráfico avanzado e informes de diferentes niveles. Que una aplicación DPI se pueda desplegar de forma generalizada en una red, proporciona una capacidad de análisis superior así como mecanismos más sencillos a la hora de implementar, actualizar, probar y adaptarlos en función de la carga de trabajo.
- Implementación coordinada de servicios de red y en la nube para empresas, permitiendo ofrecer servicios en función de sus necesidades.
- Virtualización de servicios que requieren dispositivos hardware en las

instalaciones del cliente como: cortafuegos, IDS, funciones de encaminadores, aceleradores y optimizadores WAN.

- Implementación de nodos IP que soportan por ejemplo, tecnologías como CG-NAT y BRAS en servidores estándar de gama alta, ofreciendo la oportunidad de reutilizar de manera eficaz el hardware en función de la demanda.

3.7 Futuro y Evolución

En los últimos años se ha incrementado el desarrollo de tecnologías relacionadas con las redes, como NFV y SDN, y que suponen una promesa de futuro en la carrera de la innovación tecnológica, como se puede observar en la Figura 3.5.

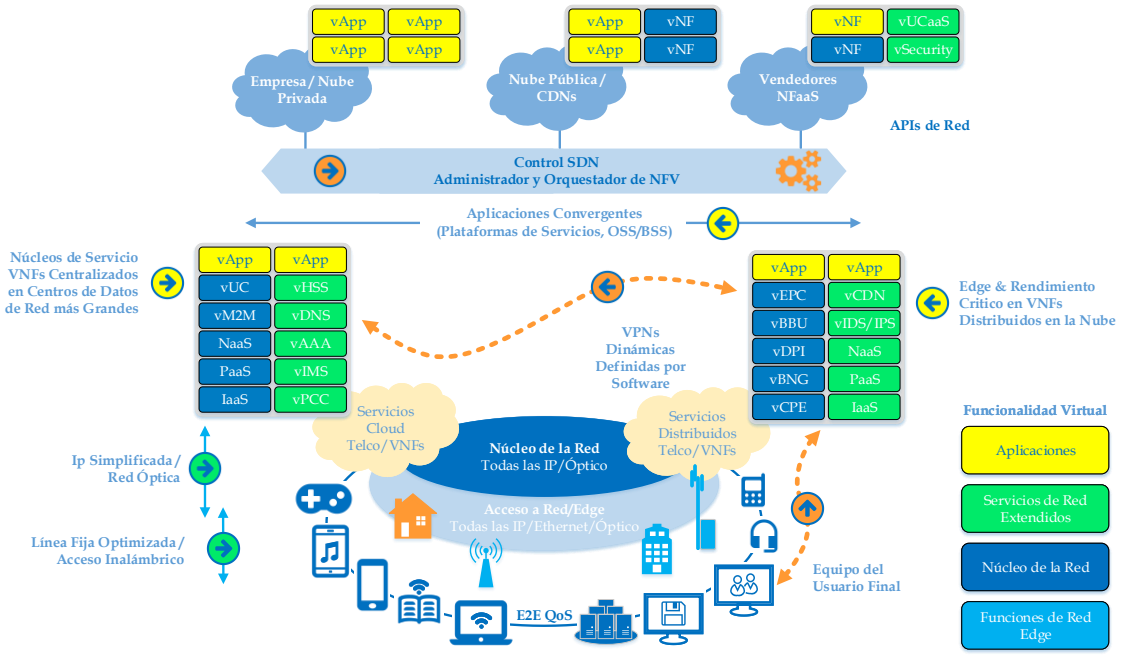

Figura 3.5: Futura arquitectura de la red de un operador con SDN y NFV. [LIT15]

Pero para que esta promesa se lleve a cabo, las empresas y los operadores de red deben [LIT15] transformar sus redes heterogéneas en un conjunto automatizado y homogéneo. Para lograr este objetivo deben mantener la

fiabilidad y la calidad de servicio que tienen en la actualidad. Las empresas deben planificar muy cuidadosamente esta transformación en sus redes, ya que este proceso necesita realizar iteraciones frecuentemente y trabajar tanto con proveedores, como con clientes líderes en su sector para maximizar el beneficio y el aprendizaje en cada fase.

Las oportunidades y decisiones a las que se enfrentan tanto las grandes empresas como los operadores de red, dada la amplitud de los cambios que suponen SDN, NFV y el mercado en la nube, hace que el mundo de las telecomunicaciones cambie por completo. Ante un cambio de tal magnitud el primero que consiga adaptarse con mayor acierto y rapidez, será el que prospere en este nuevo ecosistema.

4. ARQUITECTURA SELFNET Y SUBCAPA DE MONITORIZACIÓN

4.1 Introducción

La fabricación de dispositivos se ha visto incrementado de manera exponencial en los últimos años, y cabe esperar que lo siga haciendo en los años venideros. Esto implica el aumento de manera considerable del tráfico generado por estos dispositivos, por lo que las actuales infraestructuras de telecomunicación (4G) no serán capaces de satisfacer la demanda generada. Esto ha motivado la incipiente necesidad de promover una nueva forma de afrontar este cambio en el paradigma actual, con el consiguiente desarrollo de redes de telecomunicación de quinta generación, o las también denominadas redes 5G.

En este contexto, el proyecto SELFNET diseña e implementa un marco de gestión autónoma de la red, con el fin de lograr capacidades de auto-organización de la gestión de las infraestructuras de red. Esto se consigue [SEL] mediante la detección y mitigación automática de problemas, que actualmente se siguen resolviendo de manera manual, reduciendo así los costes y mejorando la experiencia del usuario.

SELFNET integra en su arquitectura tecnologías innovadoras tales como las Redes Definidas por Software (SDN), Virtualización de Funciones Virtuales (NFV), Redes Auto-Organizables (SON), computación de la nube, inteligencia artificial, calidad de experiencia (QoE), calidad de servicio (QoS) que proporcionarán escalabilidad, flexibilidad y una gestión eficiente de las redes y sus componentes.

SELFNET [SEL] por tanto se desmarcará de las redes tradicionales adquiriendo características como la auto-protección frente a ataques de denegación de servicio (DDoS), la auto-reparación frente a fallos dentro de la arquitectura de la

red y auto-optimización proveyendo capacidades para mejorar dinámicamente la red y la calidad de experiencia de los usuarios.

4.2 Arquitectura SELFNET [NCR15]

A continuación se explicará con detalle las diferentes capas que componen la arquitectura de SELFNET, como se puede observar en la Figura 4.1.

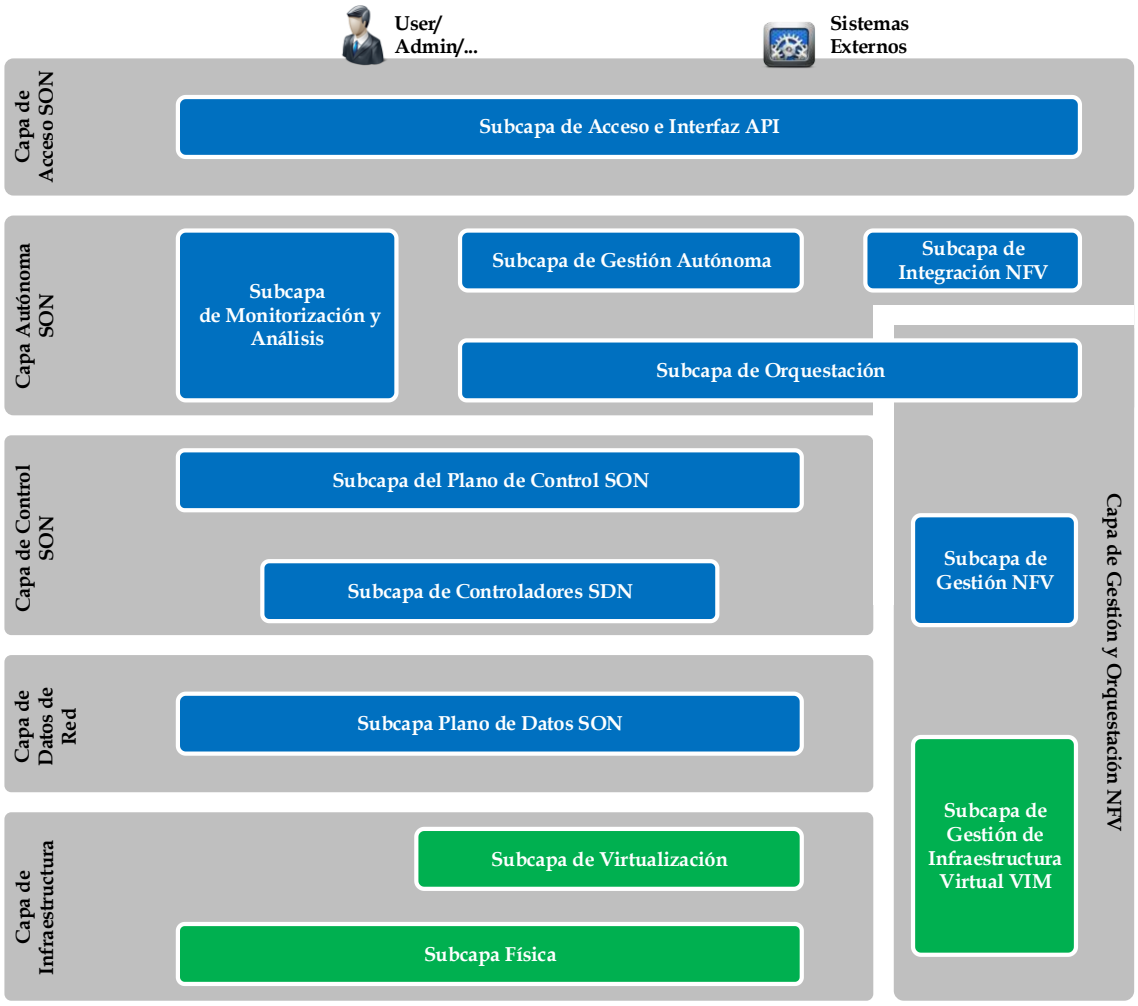

Figura 4.1: Arquitectura de SELFNET. [NCR15]

4.2.1 Capa de Infraestructura

La capa de infraestructura es compatible con la arquitectura 5G. La infraestructura de SELFNET se divide en diferentes subcapas: *Subcapa Física y*

Subcapa de Virtualización.

Subcapa Física. Se basa en los principios de "Mobile Edge Computing" (MEC). Un centro de datos principal se despliega proporcionando grandes capacidades computacionales y de comunicación, mientras que un gran número de centros de datos más pequeños, ubicados en la frontera de la red y a la vez conectados al centro de datos central, proporcionan una capacidad computacional y de comunicación más específica y reducida. Se ha definido una ubicación lógica como una manera de controlar dónde se asignan los diferentes servicios dentro de la arquitectura. Sin embargo, esta ubicación lógica puede localizarse en distintas ubicaciones físicas separadas.

Subcapa de Virtualización. Proporciona la capacidad de suministrar infraestructuras virtuales sobre las físicas, con el fin de mejorar la gestión, el aislamiento y la consolidación de los recursos computacionales. Esta subcapa se caracteriza por su interoperabilidad ya que se puede conectar y acceder a diferentes tecnologías de virtualización (hipervisores). Esta subcapa también proporciona cualidades como la confiabilidad, la seguridad y la conectividad de los servicios disponibles.

4.2.2 Capa de Datos de Red

En esta capa se instancian un gran número de funciones de red y máquinas virtuales, que a su vez están interconectadas en una topología diseñada con el fin de proporcionar las funcionalidades que requieren los usuarios. La capa de red virtualizada, se divide entre el plano de datos y el plano de control, y se encarga de proporcionar funciones de red virtualizadas. Cada aplicación se puede asignar tanto en un centro de datos como en la frontera de la red.

En el *plano de datos*, los dispositivos virtuales proporcionan servicios a lo largo de la red, a través de las máquinas virtuales donde se está ejecutando dicho dispositivo. Así mismo, en el *plano de control*, los encargados de conectar las

diferentes aplicaciones con los controladores SDN.

Por lo tanto, conceptualmente la arquitectura SELFNET ha previsto la diferenciación de al menos dos situaciones lógicas donde las aplicaciones SDN pueden ser asignadas: un controlador situado en la frontera y otro en el centro de datos, con el fin de proporcionar un mecanismo para controlar la asignación de las aplicaciones SDN dentro de las redes 5G. Vale la pena mencionar que el soporte "multi-tenancy" sobre las aplicaciones SDN que se ejecutan dentro del controlador, es una innovación que SELFNET analizará con el fin de explotar diferentes maneras de manejar la gestión de las aplicaciones SDN.

4.2.3 Capa de Control SON

La capa de control SON está compuesta por la subcapa de Control SON y la subcapa de Controladores SDN.

Subcapa de Control SON. Esta capa contiene las aplicaciones que van a permitir recolectar los datos de los sensores desplegados a través de todo el sistema (sensores SON), y las aplicaciones que serán responsables de hacer cumplir las acciones en la red (Actuadores SON), como parte de los mecanismos actuadores para proporcionar inteligencia en redes 5G. La capa de control SON debe tratar con arquitecturas de red complejas en la subcapa de funciones de datos y, al mismo tiempo, facilitar la gestión y el despliegue de nuevos servicios. Existen dos mecanismos para poder llevar a cabo estas tareas: el uso de APIs homogéneas y la abstracción de las infraestructuras para simplificar su gestión y orquestación. Una interfaz que esté basada en políticas comunes, puede combinar estos dos mecanismos integrando funciones de monitorización y control en un lenguaje común. En resumen, la finalidad de esta capa es la de traducir las políticas autónomas de toda la red en configuraciones de elementos de red específicas.

Subcapa de Controladores SDN. El uso de conmutadores para SDN es una decisión arquitectónica de la infraestructura de SELFNET, debido a que permite manejar la gestión de los servicios de red tanto en el plano de datos como en el plano de control. La combinación de conmutadores basados en software y conmutadores basados en hardware SDN, está siendo considerada dentro de la propia arquitectura de SELFNET para proporcionar más alternativas a la hora de encontrar ubicaciones más adecuadas y conseguir por tanto mayor flexibilidad a la hora de implementar nuevas aplicaciones SDN.

4.2.4 Capa Autónoma SON

La capa autónoma SON es la encargada de proporcionar la automatización de funciones en la red. La capa recopila la información de utilidad sobre el comportamiento de la red, que después utiliza para evaluar la condición de la red y realizar diagnósticos para encontrar problemas, y que posteriormente indicará como resolverlos. Esta capa se divide en cuatro subcapas principales: monitorización y análisis, gestión autónoma, orquestación e integración de VNFs.

Subcapa de Monitorización y Análisis. El objetivo principal de esta capa es proporcionar una visión general del marco que se diseñará para monitorizar y analizar el comportamiento de la red o los incidentes que ocurran en la misma. Esta subcapa de divide a su vez en tres módulos:

- *Módulo de Descubrimiento y Monitorización*: recoge y almacena todos los datos de los sensores tanto SDN como NFV.

- *Módulo de Agregación*: realiza la agregación y correlación de las métricas de bajo nivel proporcionadas por módulo de descubrimiento y monitorización.

- *Módulo de Análisis*: su objetivo es analizar la información proporcionada por el módulo de agregación, obteniendo la información necesaria para facilitar la toma de decisiones.

Subcapa de Gestión Autónoma. Esta subcapa está considerada como el cerebro de SELFNET ya que proporciona las capacidades de auto-reparación, auto-protección y auto-optimización gracias a que trata de manera proactiva y reactiva tanto los problemas existentes como los potenciales. Esta subcapa está formada por cuatro módulos: *Módulo TAL*, *Módulo de Diagnóstico*, *Módulo de Planificación de Toma de Decisiones* y *Módulo Ejecutor de Acciones*.

- *Módulo TAL*: denominado *Lenguaje Autónomo Táctico*, incluye un lenguaje autonómico y su biblioteca asociada, que definen las estrategias autonómicas dentro de SELFNET.

- *Módulo de Diagnóstico*: aprovecha la inteligencia artificial, la minería de datos y los algoritmos estocásticos para proporcionar inteligencia en el diagnostico de los problemas de red y la toma de decisiones.

- *Módulo de Planificación de Toma de Decisiones*: se encarga de decidir un conjunto de acciones correctivas y preventivas para tratar los problemas identificados y potenciales de la red.

- *Módulo Ejecutor de Acciones*: proporciona un conjunto consistente de acciones programadas para ser implementadas en la infraestructura de la red.

Subcapa de Orquestación. Se encarga del despliegue real de las aplicaciones SDN y NFV siguiendo las instrucciones especificadas por el módulo de *Planificación de Toma de Decisiones*. Esta subcapa está formada por tres módulos:

- *Módulo Orquestador*: es el encargado de realizar el despliegue de las aplicaciones e intermediar en la asignación de los recursos del sistema

para que se haga de la manera más óptima posible.

- *Módulo Gestor de Aplicaciones*: gestiona el ciclo de vida de las aplicaciones, permitiendo al *Orquestador* instalar, eliminar, actualizar, configurar, iniciar, reiniciar, detener o reanudar una aplicación seleccionada en función del plan de acción que se esté ejecutando.

- *Módulo Gestor de Recursos*: proporciona un punto de referencia para que el Orquestador interactué con el controlador de la nube y la red.

Subcapa de Integración de VNFs. Se compone de las funciones de gestión de las VNFs, mecanismos y herramientas responsables de la encapsulación de los actuadores y los sensores SELFNET en VNFs comunes y homogéneas.

4.2.5 Capa de Gestión y Orquestación NFV

Esta capa tiene la capacidad de gestionar el control tanto de las infraestructuras de cálculo, como la infraestructura de red disponibles en la arquitectura. La capa de Gestión y Orquestación NFV está compuesta por la *subcapa de Gestión NFV* y la *subcapa de Gestión de Infraestructura Virtual VIM*.

Subcapa de Gestión y Orquestación NFV. Proporciona la capacidad para desplegar nuevas aplicaciones SDN dentro del marco SELFNET, a través de una interfaz con la *subcapa de Controladores SDN*. Además, la *capa de Control SON* contribuye a esta subcapa proporcionando las aplicaciones necesarias para la supervisión de métricas de red y para la ejecución de acciones dentro de la red.

Subcapa de Gestión de Infraestructura Virtual VIM. El VIM es una plataforma que proporciona servicios de computación en la nube, donde los usuarios pueden solicitar recursos de infraestructura como almacenamiento, computación, red, etc.

4.2.6 Capa de Acceso SON

La capa de Acceso SON se encuentra en la parte superior de la arquitectura SELFNET, como se puede observar en la Figura 4.1. Es el punto intermedio donde el administrador y la plataforma de servicios convergen. Esta capa implementa una interfaz amigable y útil para administrar los servicios expuestos en SELFNET. Además, esta interfaz registra los actuadores y sensores que están desplegados en ese momento en SELFNET.

4.3 Monitorización y Descubrimiento

Entre las tareas principales [CV17] de SELFNET se encuentran la monitorización y el descubrimiento de las diferentes métricas generadas por la infraestructura virtualizada. Las métricas recogidas pueden incluir métricas de bajo nivel, indicadores clave de rendimiento (KPI), y métricas del estado de la red con el fin de obtener una información completa sobre la situación de la red.

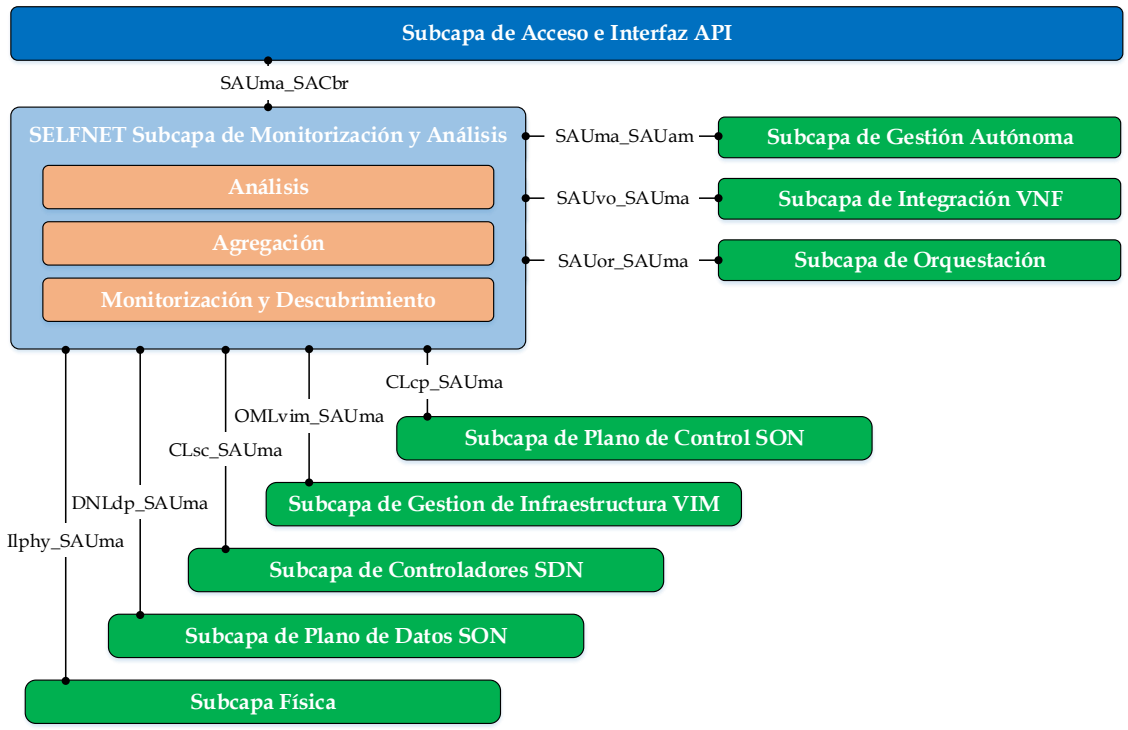

Figura 4.2: Interfaces conectadas a la Subcapa de Monitorización y Análisis. [CV17]

La gran ventaja de SELFNET frente a sistemas de monitorización tradicionales, radica en que los nodos de monitorización y la información que son capaces de capturar, se obtienen de sensores que se pueden asignar dinámicamente en la red. Sin embargo, en la monitorización tradicional, se asignan de manera estática. Gracias a esto se consigue una mayor flexibilidad y eficiencia a la hora de capturar la información generada por los sensores.

La información capturada estará a disposición de cada una de las capas que conforman la arquitectura SELFNET, a través de interfaces, como se puede observar en la Figura 4.2. En la Tabla 4.1 se pueden observar las ocho interfaces que se utilizan tanto para recopilar métricas, como para enviar los resultados de las tareas de análisis.

Nombre	Origen	Destino	Información enviada
Ilphy_SAUma	Físico	Monitorización y Análisis	Métricas Físicas
OMLvim_SAUma	VIM	Monitorización y Análisis	Métricas y Recursos Virtuales
DNLdp_SAUma	Plano de Datos SON	Monitorización y Análisis	Métricas del Plano de Datos
CLsc_SAUma	Controladores SDN	Monitorización y Análisis	Métricas de Controlador SDN
SAUvo_SAUma	Integración VNF	Monitorización y Análisis	Descripción de Sensores NFV
SAUor_SAUma	Orquestador	Monitorización y Análisis	Instanciación de Sensores NFV
SAUma_SAUam	Monitorización y Análisis	Gestión Autónoma	Resumen de alto nivel de problemas actuales y potenciales de la red
SAUma_SACbr	Monitorización y Análisis	Acceso e Interfaz API	Estado de la Monitorización y Análisis

Tabla 4.1: Interfaces de Monitorización y Análisis. [CV17]

4.3.1 Arquitectura de Alto Nivel

El objetivo principal de la *subcapa de monitorización y análisis*, es la de proporcionar un conjunto de componentes para analizar el estado de la red, mediante el uso de la información recolectada a través de los diferentes elementos convenientemente situados a lo largo de la red, como se comentó en el Capítulo 4.2.4.

Toda la arquitectura se divide en bloques funcionales, como se puede observar en la Figura 4.3. En la parte inferior, se encuentran las fuentes de datos.

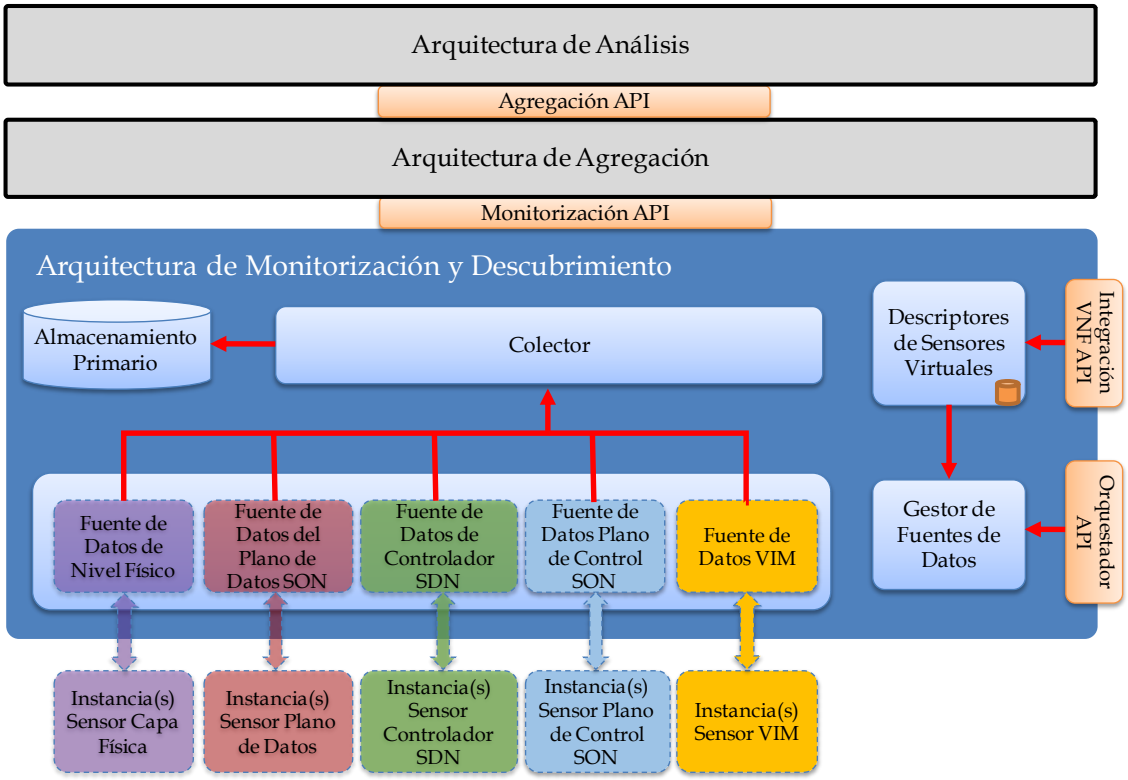

Figura 4.3: Arquitectura de Monitorización y Descubrimiento. [CV17]

Esta arquitectura permite transferir los datos de cada uno de los elementos supervisados en la red. El origen de los datos es capaz de escoger el método de comunicación, bien sea bajo petición (*polling*) o bien mediante entrega inmediata (*pushing*), para recibir los datos de los elementos supervisados.

Con el método de comunicación *polling*, el servidor que necesita los datos, envía una petición para que el elemento de la red que posee esos datos, se los entregue siempre y cuando haya una petición por parte del servidor para extraerlos. Mientras que el método *pushing*, el elemento de la red que tenga los datos, los envía según los tenga disponibles y bajo ningún tipo de petición previa.

La fuente de datos implementa la interfaz que necesita para comunicarse con el elemento supervisor. La función de descubrimiento hace referencia a la capacidad que tiene la arquitectura de detectar nuevas instancias de datos e interconectarlas para recopilar métricas.

4.3.2 Descriptores de Sensores Virtuales

La funcionalidad del Descriptor de Sensores Virtuales es la de recibir, analizar y almacenar la información sobre los tipos de sensores disponibles en la arquitectura SELFNET. Estos descriptores interactúan directamente con la *subcapa de Integración VNF* y el *Gestor de Fuentes de Datos*.

El Descriptor de Sensores Virtuales utiliza la interfaz *SAUvo_SAUma* para permitir la comunicación con la *subcapa de Integración VNF*. Mediante esta interfaz, la *subcapa de Integración VNF* es capaz de informar de la integración, actualización o eliminación de sensores en el catálogo de SELFNET. Cada vez que hay una modificación en el catálogo, es necesario notificar a todos los Descriptores de Sensores Virtuales para que actualicen sus bases de datos.

La *subcapa de Integración VNF* proporciona una cola de mensajes para difundir cualquier acción llevada a cabo en el catálogo. Los Descriptores de Sensores Virtuales tienen un cliente que se suscribe a este canal y actualiza el catálogo de sensores locales. El catálogo de sensores locales está disponible para el entorno de Monitorización y Descubrimiento, a través de la interfaz de los Descriptores de Sensores Virtuales y el Gestor de Fuentes de Datos. Este catálogo, refleja a

partir del catálogo principal de la *subcapa de Integración VNF*, la información esencial del sensor necesaria para conectarse a un sensor y recuperar los datos detectados. El catálogo no tiene información sobre la ejecución de las instancias del sensor, pero proporciona datos relacionados con las métricas del sensor y la comunicación.

4.3.3 Gestor de Fuentes de Datos

En el entorno de monitorización de SELFNET, la fuente de datos se define como un componente funcional, que sirve como interfaz entre el receptor de la información y el sensor que está siendo analizado, y se encarga de recopilar los datos de los sensores monitorizados. Por tanto, a la hora de instanciar un sensor, se necesario tener información como la MAC, la dirección IP, etc., que se obtiene en el momento de integración del sensor en el catálogo de SELFNET. Una vez que este instanciado el sensor, el *Orquestador* notificará al *Gestor de Fuentes de Datos* la información necesaria para poder configurarlo, y así poder empezar a recolectar información.

4.3.4 Instancias de Fuentes de Datos

Este apartado define las diferentes fuentes de datos y la forma en que se recogen las métricas.

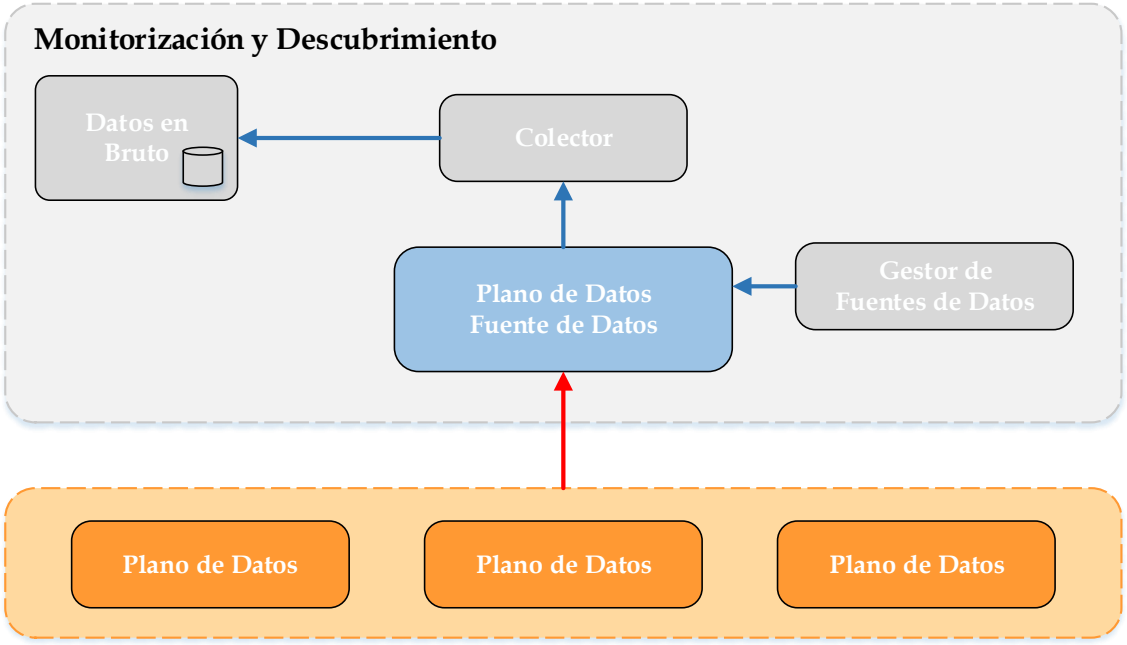

Figura 4.4: Fuente de Datos. [CV17]

Como se puede observar en la Figura 4.4, la recopilación de datos se realiza a través de una estructura común, donde las instancian envían las métricas que serán recibidas por los componentes correspondientes. Una vez haya llegado esta información, se envía a través del colector para guardar los datos en bruto.

La Tabla 4.2 resume los tipos de métricas en función del origen de la instancia.

Instancias de Fuentes de Datos	Descripción
Infraestructura Física	Métricas relacionadas con el equipo físico, que mantiene los recursos virtuales. También incluye los elementos de red que permiten la conectividad entre los diferentes componentes.
Gestor de Infraestructura Virtual	Mide los recursos virtualizados de la nube que se están ejecutando en la parte superior de la subcapa física. Incluye a los usuarios, conmutación virtual y gestión de enrutamiento y la ubicación de los recursos de computación conectados a los puertos del conmutador.
Controlador SDN	Proporciona la información relacionada con el comportamiento de la red basada en la vista del controlador SDN. En otras palabras, el controlador puede inferir las estadísticas de red basadas en la información enviada por la interfaz de plano de datos y el plano de control (Ej: Openflow).
Plano de Datos SON	Proporciona características de tráfico de red a bajo nivel. La monitorización del flujo se ha convertido en una solución apropiada. Las mediciones del flujo pueden proporcionar estadísticas de tráfico útiles con pocas mediciones.
Plano de Control SON	Proporciona la colección de métricas de los sensores que se despliegan a través de toda la infraestructura virtual. Estos sensores miden y recogen métricas físicas y/o virtuales específicas dependiendo del propósito del sensor.

Tabla 4.2: Instancias de Fuentes de Datos. [CV17]

4.3.5 Colector

El colector es un componente que se encarga de recibir toda la información que han recolectado los sensores. La recepción de esta información se hace a través de un bus, donde el colector está permanentemente escuchando. Posteriormente se transforman los datos y se ubican en una base de datos. La ventaja que tiene el colector, es que funciona independientemente del tipo de base de datos, por tanto, disfruta de gran flexibilidad a la hora de presentar la información.

El bus por donde el colector recibe los datos, está provisto de múltiples canales para el envío de datos. Estos canales reciben el nombre de "*topics*", y son capaces de enviar grandes cantidades de información con tiempos de retardo mínimos, con prestaciones como la sincronización y la capacidad de soportar eventos.

4.3.6 Almacenamiento de los Datos Primarios

Las bases de datos que contienen toda la información recibida, almacenan los datos utilizando un formato genérico. Puesto que la información tiene un tamaño desmesurado, no es posible utilizar bases de datos relacionales. Por tanto, para manejar de forma eficiente esta información, se utilizarán bases de datos NoSQL, donde el principal objetivo será el de realizar consultas en un corto periodo de tiempo con la mayor rapidez posible.

4.3.7 NorthBound API

Todos los datos almacenados son publicados en un bus de mensajes, incluido en el API de monitorización, donde pueden ser accedidos por los consumidores de SELFNET. La información almacenada en la base de datos puede ser accedida por un API REST.

5. FUENTE DE DATOS SNORT

En el capítulo anterior se menciona de forma conceptual el proceso de recolección de datos y métricas a partir de diversas *Fuentes de Datos*. En el presente capítulo se detalla el diseño e implementación de una *Fuente de Datos* basado en la arquitectura SELFNET. Además, se explica el proceso de recolección de información a partir de una *Fuente de Datos*, su procesamiento y envío al colector.

Las *Fuentes de Datos* son diversas instancias que se puedan encontrar en cualquier punto de la arquitectura SELFNET, conteniendo información de distinto tipo. Toda esta información es recogida en una plantilla previamente establecida y en un formato XML. Esta plantilla es usada en común para todas las *Fuentes de Datos* pero recoge una información específica para cada tipo de instancia.

5.1 Diseño

Las *Fuentes de Datos* son los elementos encargados de recolectar la información enviada por los sensores, normalizar y enviar dicha información al colector. Debido a que las *Fuentes de Datos* son controladas por el Gestor de *Fuentes de Datos*, es necesario que cada *Fuente de Datos* reciba una configuración inicial. Así mismo, la *Fuente de Datos* debe conectarse con el sensor para poder recibir las métricas. El presente trabajo tiene como fuente un sensor tipo SNORT [SNT] capaz de detectar alertas o potenciales amenazas en el tráfico que circula por la red.

En este sentido, se ha utilizado un diseño basado en una topología de tipo bus, el cual consta de dos bus de comunicación, estos mantienen informado en todo momento al colector, con los datos recopilados de las diversas instancias de *Fuente de Datos*. El bus utilizado tiene que ser capaz de funcionar de forma

constante, eficaz, estable y ser tolerante a fallos garantizando en todo momento la no perdida de información.

Luego de establecer el medio de comunicación, el segundo punto importante es el poder manejar los datos recibidos de los sensores. Estos datos son recogidos en formato XML y son enviados a través del bus, para luego pasar a un servidor donde van a ser procesados y filtrados según el fichero de configuración recibido por parte del *Orquestador* para su posterior envío al colector. La Figura 5.1 muestra un ejemplo de las entradas y salidas con las que el SNORTDataSource debe trabajar.

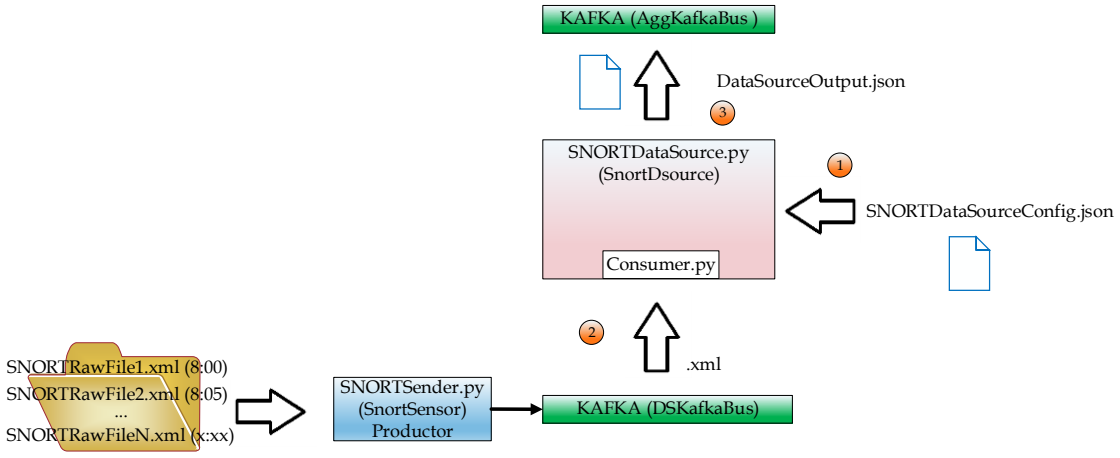

Figura 5.1: Arquitectura SNORT.

Como tercer punto se encuentra el servidor SNORTSensor, el cuál simula la carga generada por los sensores y es el encargado de inyectar ficheros XML a través del bus.

Para un mejor entendimiento del diseño realizado en este proyecto, se procede a usar diagramas de secuencias, donde se puede observar cómo funciona el sistema desde un punto más abstracto, de modo que se pueda llegar a comprender de una forma más simple, el funcionamiento de los mecanismos de envío y recepción de datos así como los pasos realizados en el proceso.

En la Figura 5.2 se puede observar la secuencia de envío de datos. En primer lugar, el script SNORTSender.py lee los parámetros de configuración. En segundo lugar, establece conexión con el bus Kafka [AK]. Tercero, verifica la conexión con Kafka. Cuarto, lee el fichero SNORTRawfile.xml(ver ANEXO A). Quinto, envía por Kafka el fichero XML. Sexto, recibe la confirmación de fichero recibido mediante una señal ACK. Séptimo, lapso de tiempo hasta recibir la segunda confirmación de mensaje consumido. Y por último Octavo, la secuencia se repite tantas veces existan mensajes por enviar.

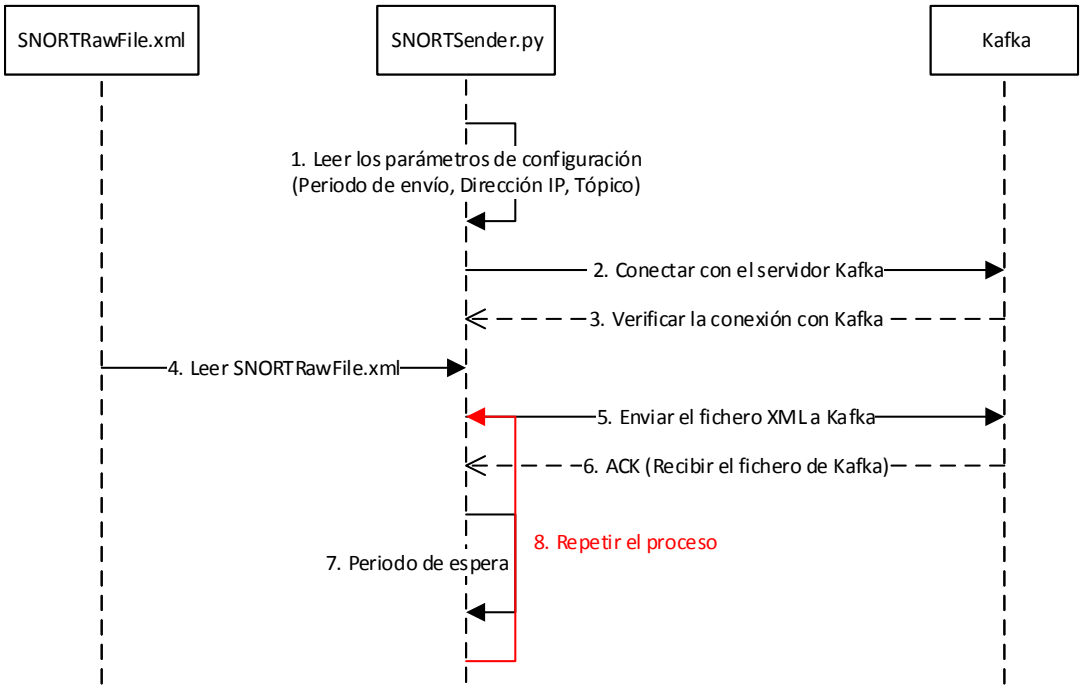

Figura 5.2: Cronograma de tiempos del SNORTSender.py

En la Figura 5.3 se puede observar la secuencia por el cual se recibe un mensaje. En primer lugar, se obtiene el contenido del fichero de configuración (ver ANEXO B). En segundo lugar, se lee los parámetros de configuración. Tercero, se establece comunicación con el bus Kafka. Cuarto, SNORTDataSource.py recibe el mensaje del bus. Quinto, se genera la salida de información. Sexto y último, se envía la salida al siguiente bus Kafka.

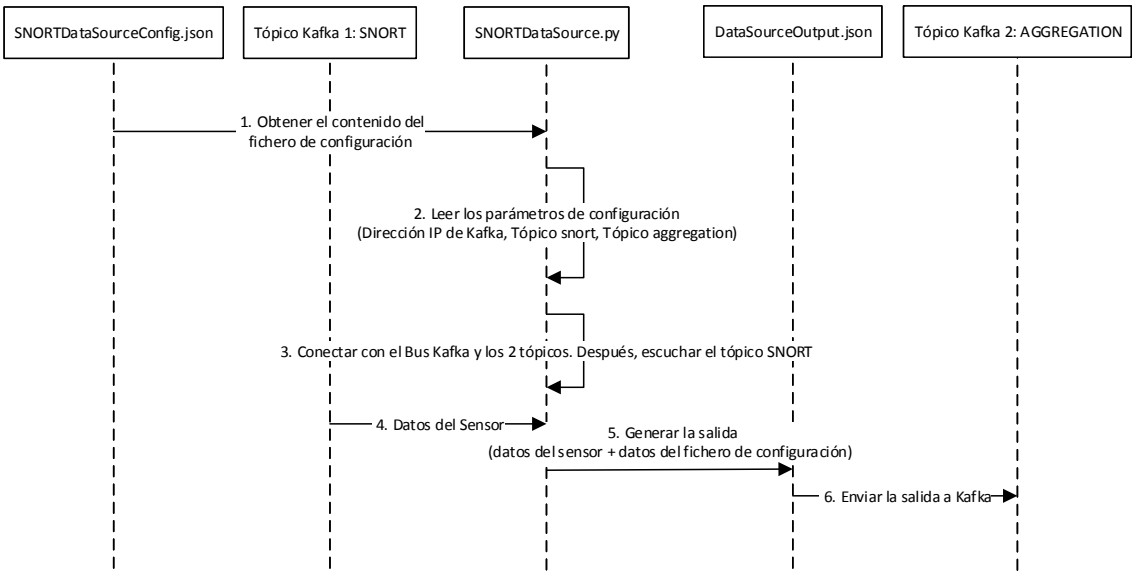

Figura 5.3: Cronograma de tiempos del SNORTDataSource.py

5.2 Implementación

En primer lugar hay que mencionar que para este proyecto se está trabajando en un entorno virtual, simulando que estamos en una red SDN y todo esto sobre un servidor físico que se encuentra ubicado en la red de la UCM. A continuación se procede a explicar de forma más detallada el proceso de implementación llevado a cabo en este proyecto.

5.2.1 Entorno de Desarrollo

El despliegue del sistema es como se muestra en la Figura 5.1. Los cuatro servidores virtuales que se crearon para la realización de este proyecto, tuvieron en un principio las mismas características tanto SW como HW. Estas máquinas tienen instalado el sistema operativo Ubuntu 14.04.4 LTS y en ellos se ha instalado el software Apache Kafka 2.11. Todas las máquinas virtuales tienen dos interfaces de red, una para la interconexión local y otra para la conexión a internet además están configuradas para que estén en la misma red local y de

esta manera estar interconectadas entre ellas con direcciones de IP fijas. También ha sido necesario instalar Python 2.7, Apache Kafka y los paquetes de Python para compatibilizar con Kafka.

En los servidores que funcionan como bus de comunicación son los que tengan iniciado los servicios de Apache Zookeeper [AZ] y Kafka. En estas máquinas se ha procurado que solo se esté ejecutando los procesos básicos y de esta forma garantizar el aprovechamiento de los recursos para los servicios de Kafka. Estos servicios estarán permanentemente gestionando la recepción y envío de mensajes según las configuraciones que se han establecido en ellas.

5.2.2 Implementación de Kafka

Para cumplir con los objetivos del proyecto era necesario encontrar un sistema de mensajería que sea capaz de intercambiar información de forma inmediata, segura, que mantenga la integridad de los datos y que sea escalable. Luego de revisar diversas tecnologías y ver cuál era la que se ajustaba mejor a las necesidades de este proyecto se decidió implementar el sistema de bus Kafka.

Un bus de comunicación suele ser de dos tipos: publicación-suscripción y colas. El problema del bus de colas es que no aceptan multi-suscripciones ya que una vez el dato es leído se borra de la cola y el bus de tipo publicación-suscripción si permite emitir datos a múltiples procesos pero no tiene forma de escalar en procesamiento ya que todos los mensajes se envían a cada suscriptor.

Apache Kafka combina las funcionalidades de los dos tipos de bus, haciendo uso del concepto de grupo de consumidores. Es decir, Kafka como suscriptor permite las emisiones de mensajes a múltiples grupos de consumidores y como una cola permite que el grupo de consumidores pueda dividir los procesamientos sobre una colección de procesos (miembro del grupo consumidor).

5.2.2.1 Características de Apache Kafka

Apache Kafka al ser un sistema de almacenamiento publicador-subscriptor, particionado y replicado es muy rápido en lecturas y escrituras y esto lo convierte en la herramienta que cumple con la necesidad del proyecto para comunicar flujos de información que se generan a gran velocidad.

Se destacan las siguientes características:

- Categoriza los mensajes en *topics*.
- Los procesos que publican se denominan *brokers* y los *susbscriptores* son los consumidores de los *topics*.
- Utiliza un protocolo propio basado en TCP y Apache Zookeeper para almacenar el estado de los *brokers*.
- Se puede programar productores-consumidores en diferentes lenguajes de programación. Para este proyecto se ha utilizado Python.
- Es escalable y tolerante a fallos.

5.2.2.2 Arquitectura de Apache Kafka

Para este proyecto no se ha utilizado una implementación en clúster de Kafka, sino un solo servidor que maneja los datos y peticiones de los *topic*. Esto implica que por cada *topic* que se ha creado, su partición en el log solo se ha distribuido en el propio servidor. Se puede observar en la Figura 5.4 la estructura de un *topic* dentro del bus.

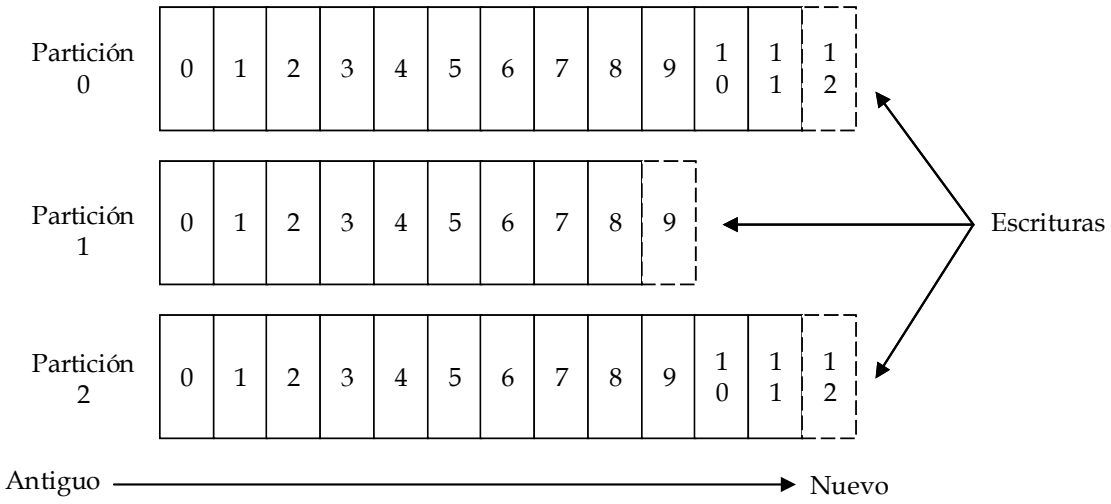

Figura 5.4: Anatomía de un *topic*. [AK]

Del mismo modo se habilita otro bus Kafka para enviar los datos ya procesados al colector.

Productor

Se ha utilizado un servidor aparte para simular los sensores, y así poder enviar los datos en formato XML al *topic* de su elección. El productor que en este caso es el sensor, será el responsable de escoger la partición adecuada para el registro que va a escribir. El productor dejará la información generada en el bus Kafka, donde posteriormente cualquier consumidor podrá tener acceso a él, como podemos observar en la Figura 5.5.

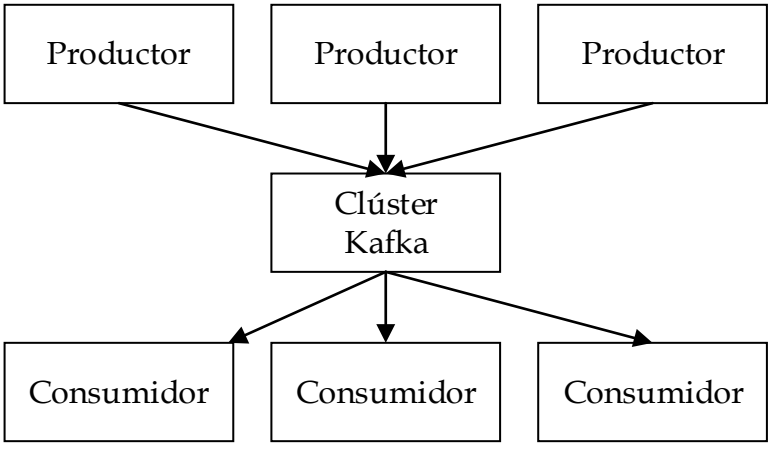

Figura 5.5: Comunicación entre Productores y Consumidores. [AK]

Consumidor

En este proyecto, el consumidor solo es ejecutado desde el servidor SNORTDataSource donde además de recibir los datos a través del bus Kafka, cumplirá la función de gestionar y filtrar la información mediante ficheros de configuración provenientes de un *Orquestador*. El proceso o procesos consumidores están permanentemente activos y a la espera de recibir algún mensaje del *topic* al cual están suscritos, como podemos observar en la Figura 5.6.

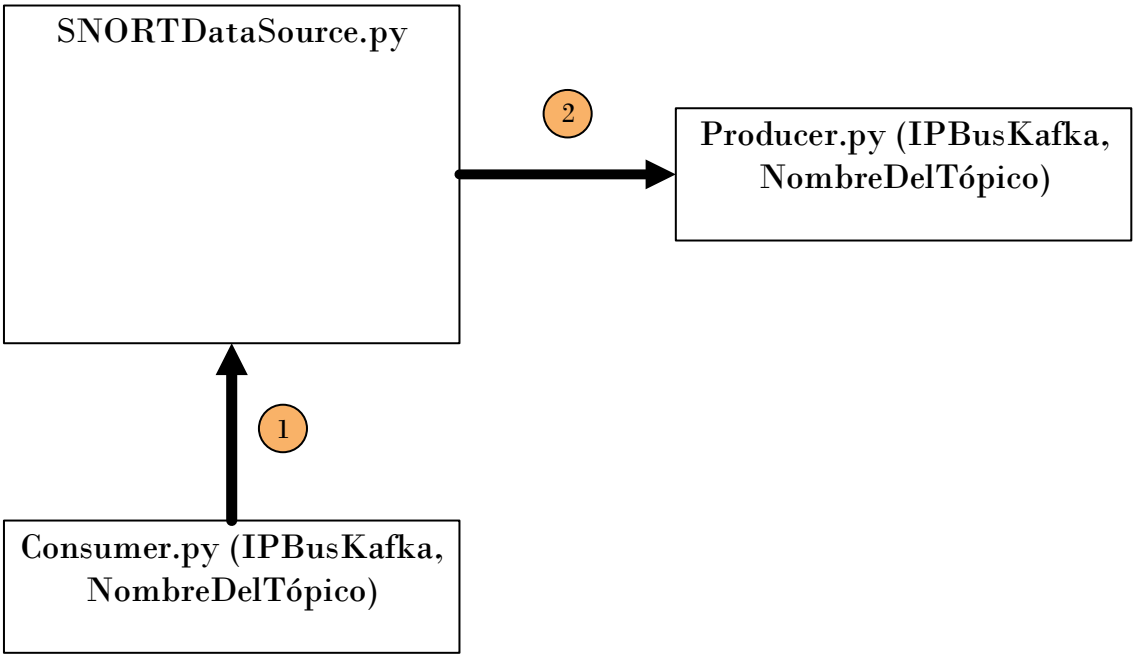

Figura 5.6: Diseño Arquitectónico.

Existen muchas posibilidades a la hora de utilizar Kafka, pues muestra una gran flexibilidad. En la Figura 5.7 se puede ver un ejemplo con otra configuración, donde hay un clúster de dos servidores Kafka que aloja cuatro particiones (P0 a P3) con dos grupos de consumidores. El grupo de consumidores A tiene dos instancias de consumidores y el grupo B tiene cuatro.

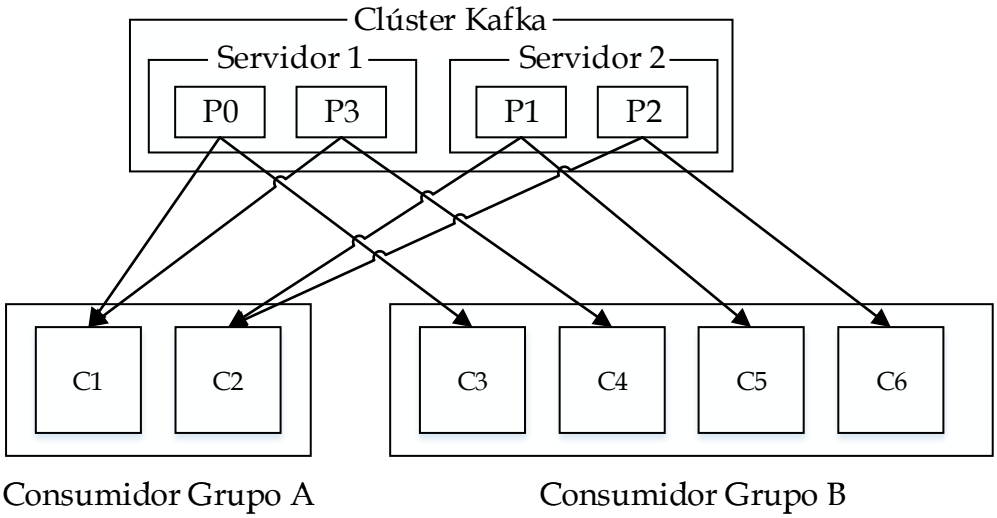

Figura 5.7: Clúster Kafka de dos servidores y dos grupos de consumidores. [AK]

Un punto que es importante al mencionar Kafka, es el seguimiento que hace sobre lo que ha consumido gracias al sistema de mensajería que implementa confirmaciones ACK. Cuando envía un mensaje y se devuelve la respuesta de recibido, el mensaje se marcará como enviado y no como consumido. Solo pasará como estado consumido cuando el consumidor haya enviado un ACK de procesado. Si el consumidor falla al enviar el mensaje de consumido, entonces el mensaje se consumirá dos veces. Kafka guarda la posición del offset del consumidor con lo cual le permite reprocesar los datos.

5.2.2.3 Funcionamiento

En este punto se procede a explicar la puesta en marcha del sistema de bus Kafka utilizada en este proyecto.

Una vez instalado Apache Kafka en el servidor con sistema operativo Ubuntu, es necesario otorgar permisos de ejecución sobre los ficheros de la carpeta "/kafka/bin".

A continuación hay que configurar Kafka y para ello hay que dirigirse al fichero "/kafka/config/server.properties".

Hay que cambiar la opción "delete.topic.enable=true" por si quieres rotar la información del *topic* de forma manual. Esta configuración se implantó para que un script limpiase el histórico del *topic* una vez al día a través de una tarea automática. El motivo era por mantenimiento y control de espacio en el servidor.

A continuación se procede a configurar el servidor socket: "listeners = PLAINTEXT: //host_name:port" (configuramos la IP del servidor y puerto 9092). De igual forma hay que configurar "Zookeeper.connect = localhost: 2181" con la IP del servidor.

Opcionalmente se puede indicar la ruta donde se guarda la información del *topic*. Hay que modificar la línea "log.dirs=/". Para este proyecto lo ubicamos en una ruta especifica con la finalidad de tenerlo monitorizado.

Kafka necesita Zookeeper para trabajar. Luego hay que ejecutar el script que inicia una instancia y se ejecuta de la siguiente manera:

- "bin/zookeeper-server-start.shconfig/zookeeper.properties".

Luego en otra consola se inicia Kafka de la siguiente forma:

- "bin/Kafka-server-start.shconfig/server.properties".

Kafka dispone de un API Python para construir productores y consumidores de mensajes. Para el productor en realidad solo hay que indicar el servidor donde se está ejecutando Kafka y el *topic* por el que escribimos los mensajes, aunque también cuenta con más parámetros para el envío de mensajes. A continuación, se detalla en el Algoritmo 1 el modulo productor usado por este proyecto.

Algoritmo 1: Productor

Input: Ruta del Fichero
Output: Mensaje asíncrona a un topic

```
    def getFichero(fichero):
        Obtiene el fichero y se guarda la información en una variable.
        return variable


    def productor(fichero):
        produce = Kafkaproducer (bootstrap_server=IP address:puerto, value_serializer = valor
para la interfaz)
        getFichero()
        Se asocia el contenido del fichero con el topic y se envía al bus.
```

Mediante la API que nos proporciona Kafka para Python, construimos el siguiente consumidor que se puede observar en el Algoritmo 2.

Algoritmo 2: Consumidor

Input: Mensaje del *topic*
Output: ack de confirmación de recibido y consumido

```
    def consumidor():
        consumer=KafkaConsumer(IP address:puerto)
        Se asocia el mensaje con el topic específico.
        for message in consumer:
            Se obtiene el mensaje
```

Los ficheros de configuración de Zookeeper como de Kafka se encuentran en el directorio "/Kafka/config/", en este fichero se puede configurar los puertos de escucha, directorio de almacenamiento por defecto, número de particiones por defecto, etc.

Zookeeper escucha en el puerto 2181 y Kafka escucha en el puerto 9092.

5.2.2.4 Entorno

El entorno donde se configuró el bus Kafka tiene las siguientes características:

- Qemu virtual CPU v.2 x2.

- 4Gb RAM.
- 8 Gb de disco.
- Sistema Operativo: Ubuntu 14.04 LTS 64-bit.

5.2.3 Implementación del SnortDataSource

Una vez implementado el bus Kafka, el siguiente paso fue implementar el servidor que aloja el consumidor además de procesar la información. El consumidor se suscribirá a uno o más *topics* para procesar el *stream* de registros que recibe del bus Kafka. Una vez recibido el mensaje, el propio consumidor deberá deserializar, tratar, filtrar, serializar y producir un nuevo mensaje en formato JSON que es enviada a través de otro bus Kafka al colector. Para lograr este cometido es necesario integrar en el consumidor Kafka las tareas de gestión de datos que a continuación pasamos a detallar.

En primer lugar hay que modificar el fichero "server.properties" con los parámetros de la propia máquina, es decir, cambiando la dirección "localhost" por la dirección IP, sin llegar a ejecutar ningún servicio de Kafka. Hecho este cambio se procede a realizar un test para comprobar el establecimiento de la comunicación con el bus y la recepción de mensajes. Para la realización de esta prueba se procede a ejecutar el siguiente comando por consola: "bin/kafka-console-sonsumer.sh –Zookeeper dirección IP:2181 –topic nombre del *topic*". Una vez establecida la conexión con el bus Kafka se comprueba por consola la recepción de un mensaje publicado por el consumidor al *topic*.

Una vez comprobado la recepción del mensaje a través de Kafka, se procede a generar el código del consumidor, usando para ello el lenguaje de programación Python y el API que proporciona Kafka para su implementación. Además se procede a integrar al código del consumidor la capacidad de gestionar la información para luego ser enviada al colector antes de procesar el

siguiente mensaje.

5.2.3.1 Funcionamiento

En esta sección se explica el proceso por el cual se gestiona la información. Como ya se había mencionado anteriormente es necesario establecer un consumidor que se encuentre permanentemente escuchando la llegada de un mensaje por el bus Kafka. Partiendo de esta base se estructura el código como se puede observar en el Algoritmo 3.

Algoritmo 3: SnortDataSource

Input: Mensaje del topic
Output: ack de confirmación de recibido y consumido

```
def Consumer():
    Obtiene el fichero JSON y se carga la configuración
    consumer = KafkaConsumer(IP address:9092, earliest, -1)
    Se asocia los mensajes con el topic específico

for message in consumer:
    Se decodifica el mensaje recibido y se guarda en XML
    snortAggregation (new_path, configFile)
```

Una vez consumido el *stream*, este es decodificado y almacenado en un fichero XML provisionalmente en un fichero temporal. A continuación la función *SnortAggregation* recibe como parámetro de entrada el fichero XML generado anteriormente, el cual contiene la plantilla con los datos proveniente de las diversas instancias de fuentes de datos. El otro parámetro de entrada es el fichero de configuración que está en formato JSON, el cual contiene los datos que hay que obtener del fichero XML.

Una vez obtenido los parámetros de entrada en la función *SnortAggregation, el* proceso empieza por crear una estructura en árbol a partir de la información obtenida del fichero XML, luego con la información obtenida del fichero de configuración en JSON se procede a guardar los diccionarios en listas. A

continuación se recorre el árbol filtrando los datos a partir de las claves obtenidas del fichero JSON. Una vez finalizado el recorrido de la estructura se captura la fecha y hora del sistema para luego estructurar un nuevo diccionario con la información filtrada y así generar un nuevo fichero JSON. Este fichero es a continuación codificado y luego publicado por un productor Kafka como *streams* al *topic* y bus definido. Para finalizar la función se procede a eliminar los ficheros XML y JSON de la carpeta temporal generados en el proceso.

5.2.3.2 Entorno

La configuración del servidor SnortDataSource se ha realizado en el siguiente entorno:

- Qemu virtual CPU x4
- 8Gb RAM.
- 8 Gb de disco.
- Sistema Operativo: Ubuntu 14.04 LTS.

Para alcanzar la funcionalidad deseada se parte de las siguientes tecnologías para los diferentes módulos del sistema:

- Python 2.7.
- Java JDK 1.7.0_121.
- Apache Kafka 2.11.
- Paquetes Python-Kafka.
- Análisis de la información: Numpy, Pandas, Matplotlib.
- Herramientas de visualización: Python-Tk, Highcharts.

6. PRUEBAS Y RESULTADOS

Una vez llevado a cabo la implementación del proyecto y siguiendo el diseño visto en el capitulo anterior, se procede a mostrar la validez del mismo. Para ello y a efectos de prueba se utilizará un conjunto de 33.787 ficheros .XML como ejemplos de salidas, recopilados por sensores basados en Solaris BSM (Basic Security Mode) de tipo host, para una red compuesta de 4 subredes y 254 host en cada una de ellas. Cada fichero es una alerta .XML. Cada alerta .XML representa una sesión de red parte de un ataque de DDoS.

Hay que destacar que estos archivos de datos fueron recolectados en un lapso de aproximadamente 3 horas y forman parte de un conjunto de datos de ejemplos de un escenario de ataque DDoS a DARPA definido por el MIT en el año 2000 [SD].

6.1 Componentes

El entorno donde se han desarrollado las pruebas está compuesto por máquinas virtuales alojadas en un mismo servidor propiedad del grupo GASS-UCM.

Servidor

El servidor que se ha utilizado para alojar las cuatro máquinas virtuales utilizadas en este proyecto, cuenta con las características que se puede observar en la Tabla 6.1.

Servidor	Características
ThinkServer TD350	Intel® Xeon® Serie E5-2600 v3 de 16 núcleos
	DDR4 de 512 GB a 2133 MHz. 16 ranuras (RDIMM/LRDIMM)
	Adaptador ThinkServer RAID 720i AnyRAID (0/1/10/5/50/6/60)
	Dimensiones: 251 mm x 459 mm x 686 mm (9,9" x 18,1" x 27,0")

Tabla 6.1: Características del Servidor

Máquinas virtuales

SNORTSender: máquina que se encarga de simular un sensor que envía información a través del bus, generando volumen de tráfico.

DSKafkaBus: máquina que hace de bus Kafka que actúa como un procesador de *streams*, consumiendo un *stream* de entrada, procesarlo y producir un *stream* de salida para uno o varios *topics*.

SNORTDataSource: máquina que se encarga de ejecutar el consumidor Kafka y a su vez gestiona el manejo de los mensajes para luego ser enviados al siguiente bus. También realiza la función de monitorización y almacenamiento de información en dataset con formato CSV.

AggKafkaBus: misma definición que el *DSKafkaBus* y que además envía el mensaje al colector.

Todas las máquinas virtuales excepto la SNORTDataSource, presentan las mismas características y el mismo software, como se puede observar en la Tabla 6.2. La máquina SNORTDataSource cuenta con el doble de recursos de memoria RAM que el resto de máquinas virtuales, y en ella se han instalado los paquetes Python y de Ubuntu necesarios para que pueda realizar unas tareas especificas. Para preparar el entorno de ejecución se ha utilizado las características que se pueden observar en la Tabla 6.2.

	Características	**Software Instalado**	**Dir. IP en servidor**
SNORTSender	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU	Apache Kafka 2.11 Python 2.7	192.168.122.164/24
DSKafkaBus	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU	Apache Kafka 2.11 Python 2.7	192.168.122.65/24
SNORTDataSource	Linux Ubuntu 64 bits, 8192 memoria base, 4 CPU	Apache Kafka 2.11 Python 2.7	192.168.122.66/24
AggKafkaBus	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU	Apache Kafka 2.11 Python 2.7	192.168.122.62/24

Tabla 6.2: Características de las Máquinas Virtuales Utilizadas.

6.2 Experimentos

El objetivo principal que se busca en este proyecto, es la de demostrar el rendimiento, la fiabilidad y la capacidad de procesamiento de datos del sistema implementado para la transmisión y gestión de mensajes desde diversas *Fuente de Datos* hasta su destino. Para ello es necesario obtener métricas en el transcurso del proceso y saber cómo de bien se están realizando los trabajos, para ello se ha implementado en el código del proyecto, funciones que van a recoger las medidas necesarias para la evaluación del sistema y almacenadas en datasets en formato CSV.

Con el propósito de evaluar el sistema, se ha definido un entorno de pruebas que simula un entorno real, como ya habíamos comentado antes el conjunto de componentes hardware y software, ha sido definido mediante máquinas virtuales.

A pesar de haber cambiado la distribución de las funcionalidades de las máquinas, en varias ocasiones, a lo largo del proyecto siempre se ha conservado la idea original de la topología. El diseño final está compuesto por 4 servidores, los cuales dos actúan como servidores de procesamiento de datos y dos como bus de tipo servidor.

Para los experimentos se ha definido dos escenarios, en el primer escenario consta en que todos los servidores cuenten con las mismas características SW y HW, para el segundo escenario se procede a aumentar la capacidad del servidor SNORTDataSource en memoria RAM y número de núcleos. Para empezar la evaluación se procede a realizar los siguientes pasos:

En primer lugar, se inicia los servicios de Apache Kafka mediante el script lanzakafka.sh diseñado para este cometido en los servidores-bus. Este script escrito en bash y Python carga los parámetros necesarios con los que se desea iniciar los servicios de Zookeeper y Kafka, ya que dependiendo de los

parámetros introducidos inicia de una forma u otra.

Luego, se ejecuta en el servidor SNORTDataSource desde la consola Python main.py, el cual inicia el entorno gráfico desde el cual se puede iniciar el consumidor o detenerlo.

A continuación se ejecuta el script sentData.sh el cual carga los ficheros XML de ejemplo desde la carpeta *Outbox* para pasarlo como parámetro al script snortsender.py el cual simula a un sensor que publica *streams* al *topic* predefinido en el bus Kafka.

6.2.1 Topología

La topología utilizada en este proyecto es de tipo bus y consta de 4 máquinas virtuales, dos bus de tipo servidor y dos servidores para procesar los datos. Todos los equipos se encuentran configurados con una dirección IP v.4 que se encuentran en la misma red local. Su distribución se especifica en la Figura 6.1.

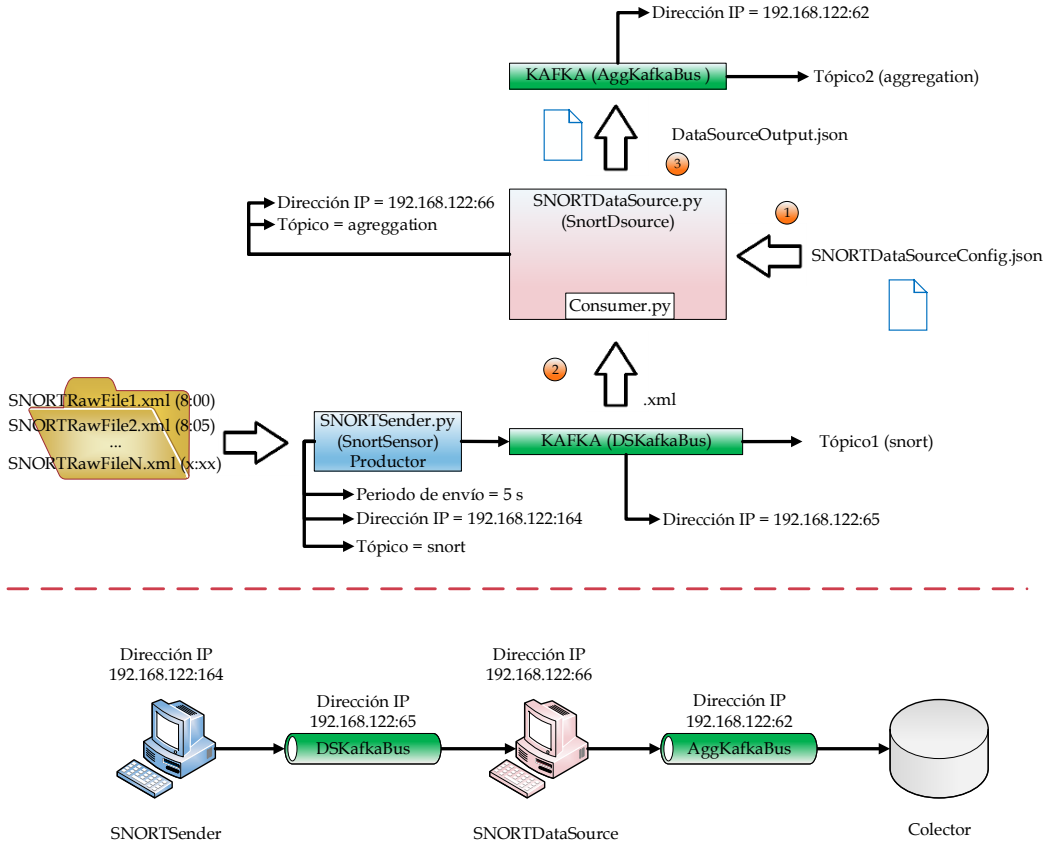

Figura 6.1: Implementación del Diseño Propuesto.

6.2.2 Escenario 1

En este primer escenario se busca comprobar el funcionamiento del sistema, usando para este fin la topología de bus antes mencionada y una configuración HW en común para todos los servidores. Para esta puesta en marcha se busca una topología sin problemas de retardos, saturaciones, cuellos de botella o pérdidas de conexión.

La evaluación en el primer escenario, ha consistido en la obtención de métricas, como por ejemplo el tiempo en que se procesa una cantidad de N datos (ficheros XML) recibidos en el servidor SNORTDataSource. Estos primeros resultados llevaron a mejorar la implementación original ocasionando mejoras en el algoritmo.

La medición empieza desde que el consumidor suscrito a un *topic* en el bus Kafka recibe un *stream* hasta que se genera un nuevo *stream* con la nueva información y es publicada al siguiente bus. Para este cometido se ha usado el concepto de monitorización pasiva, es decir, el propio sistema o aplicación recoge la información. En este caso se ha modificado el código para que recoja la información en cada recepción de mensajes, almacenando el número de paquetes recibidos en un determinado momento (medido en microsegundos).

Además se ha monitorizado los datos de los ficheros XML entrantes como por ejemplo: dirección IP fuente, puerto fuente, dirección IP destino, puerto destino y tipo de protocolo. Toda esta información se guarda en datasets en formato CSV.

Para el análisis Operacional utilizado en los escenarios, se parte de la información almacenada en los datasets obtenidos como parte de la monitorización. Esta información es utilizada para obtener métricas, mediante cálculos y operaciones se obtienen resultados que permitan evaluar el rendimiento, la productividad, etc. También nos permite contrastar los resultados a medida que se van implementando mejoras en el sistema siendo un indicativo de cómo se van desarrollando las cosas y también como una herramienta de decisión.

Para las pruebas realizadas en el escenario 1 se ha contrastado el número de trabajos realizados (2943 ficheros .XML) por unidad de tiempo (expresado en segundos). Además cabe mencionar que para este escenario las mejoras implementadas fueron solo a nivel software.

En la Figura 6.2 se contrasta los diversos resultados durante el escenario 1. La Fase 1, muestra el primer resultado obtenido en la primera implantación, obteniendo un bajo rendimiento. En la Fase 2, se obtiene un mejor resultado tras mejoraras del código. En la Fase 3, 4 y 5, las mejoras de rendimiento son mínimas.

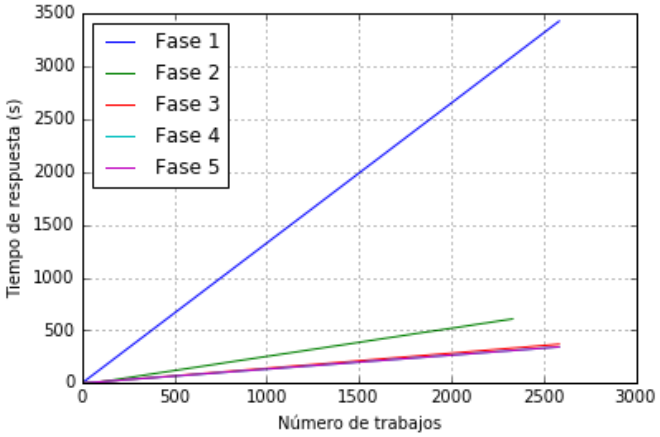

Figura 6.2: Fases en el Rendimiento Obtenido en el Escenario 1.

6.2.3 Escenario 2

Luego de evaluar en varias ocasiones el rendimiento del sistema y de implementar mejoras en el algoritmo del consumidor, se procede a implantar nuevos cambios a nivel HW en busca de mejorar el rendimiento.

En este escenario, se busca mejorar el tiempo en que se procesan los mensajes recibidos desde el bus Kafka, para ello se procede a cambiar la configuración del servidor SNORTDataSource. Este cambio consiste en incrementar la memoria RAM de 4Gb a 8Gb y el número de núcleos de 2 a 4. También se realizan mejoras en el algoritmo del consumidor, suprimiendo clases innecesarias, limpiando el código y evitando redundancia de código.

En la Figura 6.3 se muestra un contraste en el rendimiento entre los Escenario 1 y el Escenario 2, tras mejorar los recursos HW en el servidor SNORTDataSource.

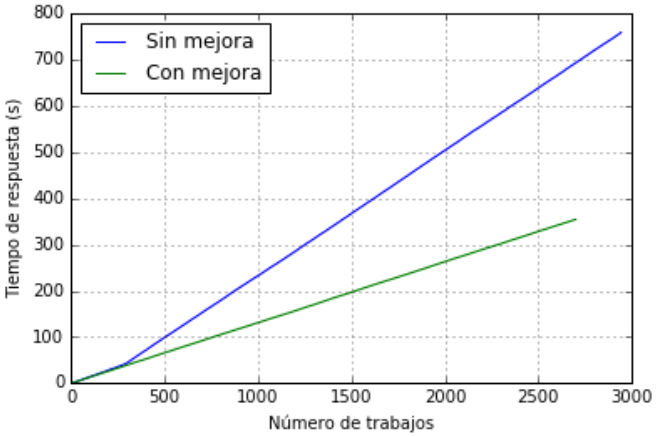

Figura 6.3: Diferencia entre Rendimientos tras la Mejora Obtenida en el Escenario 2.

Como se puede observar en la Figura 6.3 se incrementa el rendimiento al doble de lo que se tenía en el escenario 1.

En la Figura 6.4 se muestra la productividad obtenida como resultado de una evaluación realizada con una carga de 3 mil ficheros XML. Como se puede observar en la Figura 6.4 A, se agrupa el número de trabajos procesados por minuto, obteniendo en el rango de observación de los primeros 4 minutos, un máximo de 460 trabajos procesados por minuto, permaneciendo estable durante toda la prueba. De igual manera, en la Figura 6.4 B se muestra el tiempo que tarda un trabajo en procesarse al cabo de toda la evaluación. Se puede observar dos tendencias de tiempo, entre los 130 y 170 milisegundos en realizar una tarea.

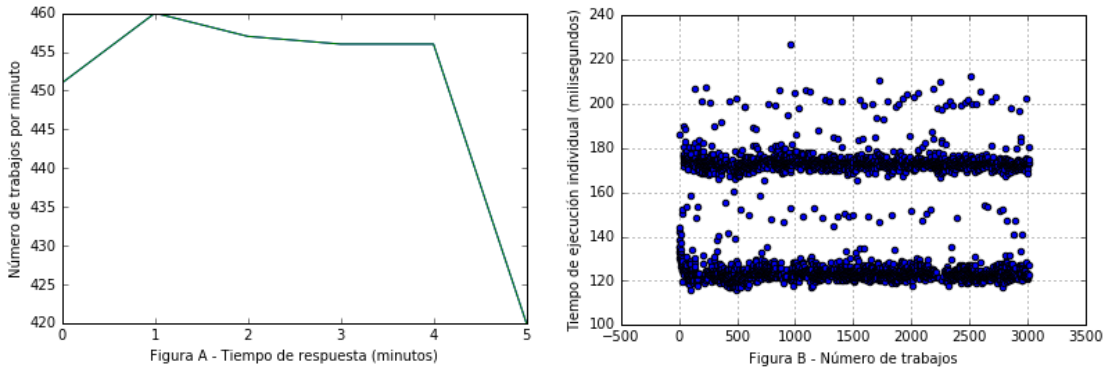

Figura 6.4: Productividad Obtenida en el Escenario 2.

En la Figura 6.5 se muestra los resultados obtenidos de medir el porcentaje de uso de CPU y de la memoria. En la Figura 6.5 A se observa que el porcentaje de uso de la memoria se mantiene en los rangos de 34 y 35 porciento de uso, permaneciendo estable hasta finalizado la evaluación. A su vez en la Figura 6.5 B, se observa el porcentaje de uso de la CPU, el cual muestra una tendencia del 5% de utilización de sus recursos durante la evaluación, mostrando niveles aceptables de uso tanto de memoria como de CPU.

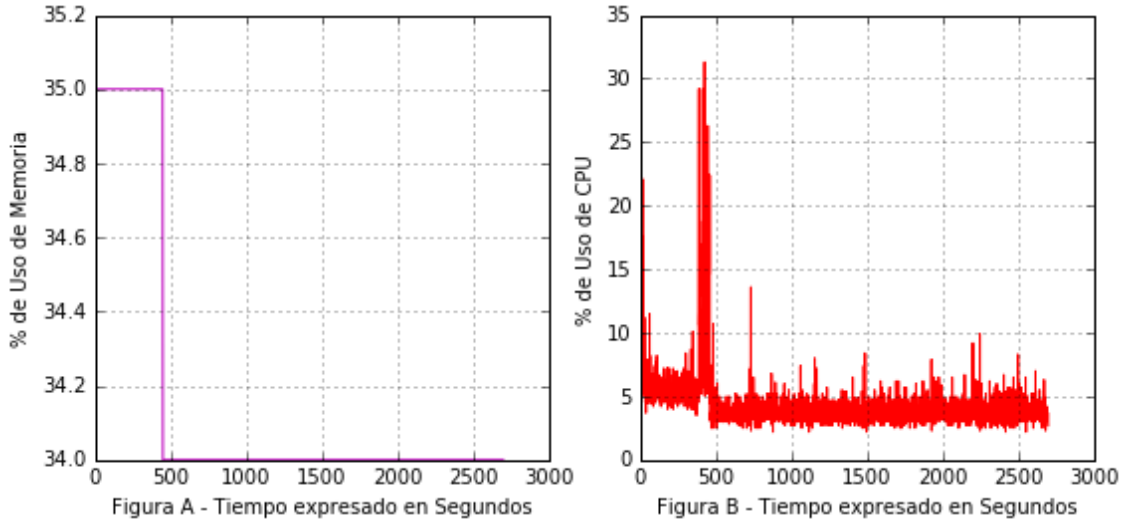

Figura 6.5: Porcentaje de uso de CPU y Memoria durante la ejecución del consumidor Kafka.

En la Tabla 6.3 se puede observar la comparación entre los escenarios 1 y 2.

	Nº de Trabajos	**Media por Fichero (ms)**	**Tiempo Máximo (ms)**	**Tiempo Mínimo (ms)**	**Uso memoria (%)**
Escenario1	3014	551.066	999.467	16.955	24
Escenario2	3014	147.421	226.722	116.051	34

Tabla 6.3: Comparación de métricas entre escenarios 1 y 2.

Para la siguiente evaluación se ha utilizado 3 productores, 3 consumidores y un *topic*. Para cada carga de trabajo se ha utilizado 2000 ficheros XML es decir un total de 6000 mil ficheros. Para esta evaluación también se mide el porcentaje de la CPU y memoria. En la Figura 6.6 se puede observar la ejecución de 3

productores Kafka en el servidor SNORTSender.

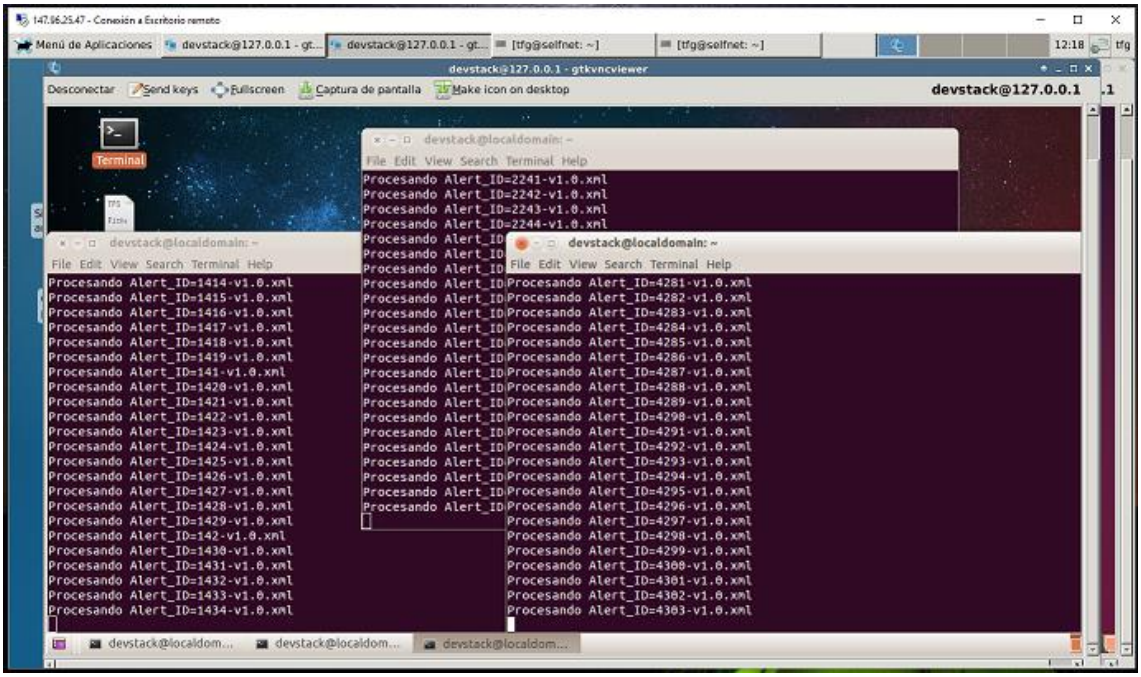

Figura 6.6: Ejecución de tres productores Kafka.

En la Figura 6.7 A, se puede observar un total de 6000 trabajos realizados en 800 segundos, si comparamos el resultado de la prueba anterior (Figura 6.3), los resultados son similares obteniendo una media de 2500 trabajos en 300 segundos. Mostrando solo un incremento del tiempo de forma directamente proporcional al número de trabajos, manteniéndose el ángulo de crecimiento para este experimento. Como se puede observar en las Figuras 6.7 B un porcentaje de uso de la memoria de forma estable, con un 21 porciento constante durante toda la evaluación. De igual forma en la Figura 6.7 C se observa que, la CPU incrementa su uso ligeramente a comparación de la prueba anterior (Figura 6.5 B) mostrando unos niveles superiores al 5% y con ciertos picos del 50% en momentos puntuales de la prueba.

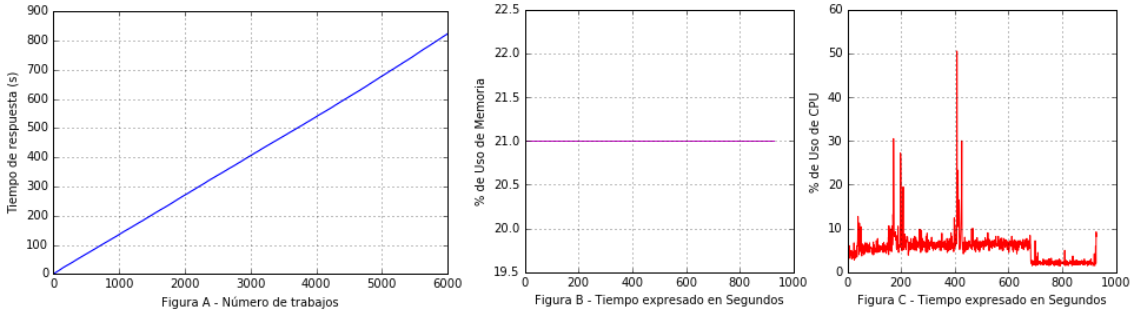

Figura 6.7: Gráficas de Rendimiento y Consumo.

Para la siguiente evaluación se procede a agregar dos *topic* en el DSKafkaBus, dos productores en el SNORTSender que apuntan cada uno a un *topic* diferente y dos consumidores en el SNORTDataSource para cada *topic*. En esta ocasión se utiliza 2 mil ficheros XML que son enviados desde el servidor SNORTSender a cada *topic* configurado en el bus. Luego de modificar los consumidores para que estén suscritos a cada topic, se procede a enviar la carga de los 4 mil mensajes a la vez.

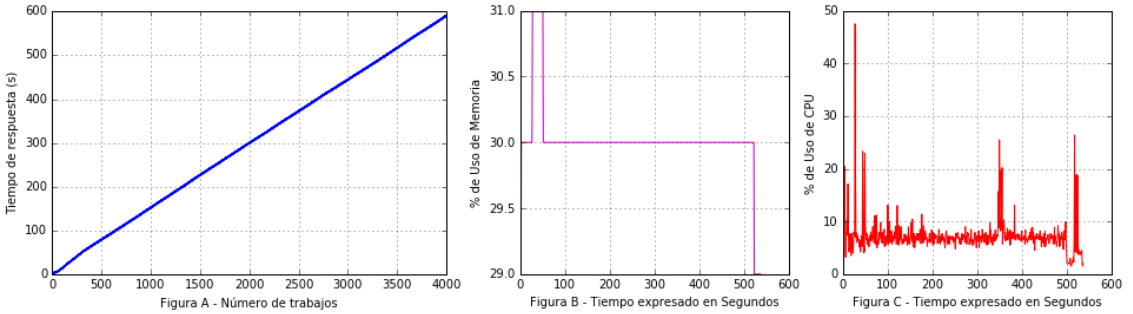

Figura 6.8: Gráficas de Rendimiento y Consumo con dos *topics*.

En la Figura 6.8 A se puede observar que el rendimiento obtenido sigue siendo similar al resto de pruebas hechas anteriormente en el escenario 2. Se observa que con 4 mil trabajos realizados en 600 segundos se asemeja a las cifras obtenidas en la Figura 6.7 A. Luego en la Figura 6.8 B se muestra un incremento del 10% de uso de la memoria con respecto al experimento anterior, moviéndose en unos rangos entre 30 y 31 porciento de uso. Esto es entendible ya que Kafka está utilizando dos colas de trabajo y por ello requiere más

recursos de memoria. De igual manera que en anteriores resultados se observa en la Figura 6.8 C un mínimo incremento en el uso de CPU, manteniéndose casi siempre entre los valores entre 5 y 10 porciento de uso.

7. CONCLUSIONES Y TRABAJO FUTURO

7.1 Conclusiones

En el presente proyecto, se ha diseñado una *Fuente de Datos* tipo SNORT basado en la arquitectura SELFNET. La fuente de datos tiene la capacidad de recolectar información de sensores virtuales y enviarlas al colector. Al mismo tiempo, se normalizan las métricas a un formato genérico capaz y se envía los resultados al colector.

Para llegar a esta meta se tuvieron que realizar varias evaluaciones hasta encontrar el equilibrio entre fiabilidad y rendimiento. Esto se ha conseguido en gran parte al uso de Kafka que hace la tarea de mensajería, almacenamiento y procesamiento de *streams*, aunque su punto fuerte es la redundancia de errores gracias al manejo de la posición del consumidor (offset), evitando posibles fallos al procesar el mensaje o perdidas de conexión, ya que al presentarse alguno de estos problemas Kafka reenviaría nuevamente el mensaje.

Una vez conseguido resolver el problema de la redundancia de errores, se centro en mejorar el rendimiento de procesado de datos y tras varias evaluaciones y luego de haber contrastado resultados, se puede asegurar que para conseguir un mayor rendimiento y productividad en la transferencia de datos es imprescindible contar con los recursos hardware necesarios para un procesamiento más rápido y eficaz.

Por tanto, para finalizar se puede concluir que el incremento de datos analizados por unidad de tiempo, es debido a la mejora en el HW de la máquina que ha procesado los mensajes (en este caso es el servidor SNORTDataSource). Si se pretende un mayor rendimiento es necesario también implementar más servidores bus en paralelo, ya que las pruebas realizadas en los escenarios 1 y 2 se ha observado que el rendimiento de la *Fuente de Datos*

SNORT tiene en promedio de 9 trabajos procesados por segundo. Así mismo, el consumo de CPU y Memoria muestran una carga media de 7 y 25 porciento de uso respectivamente. Para finalizar se concluye que el rendimiento, no mejora por mas *topics o* consumidores se aumente, sino que depende en gran medida de las mejoras en recursos hardware que se puedan implementar.

7.2 Trabajo Futuro

En el marco del diseño de un sistema de intercambio de información, existe infinidad de posibilidades de optimización en todos los aspectos del sistema. Entre los trabajos futuros se incluye la implementación de Kafka en un clúster de servidores para mejorar el rendimiento con el uso de varios *topics*, probar la opción de balanceo de información en Kafka y así evitar posibles caídas de los servidores y mejorar la redundancia de errores al tener el mismo *topic* en varios servidores.

Además, gracias a las métricas generadas por el proyecto en la fase de evaluación, se ha recopilado datos enviados por los sensores que fueron descartados a la hora de enviar los ficheros de configuración al colector, esto puede servir para que futuros proyectos puedan recolectar datasets o usar los generados por este proyecto para hacer evaluaciones de la red o utilizar los datasets para múltiples propósitos tales como: el filtrado de paquetes en una red convencional o SDN, la toma de decisión a la hora de cambiar la topologías o simplemente para hacer análisis predictivos usando el Big Data.

8. DIVISIÓN DEL TRABAJO

8.1 Daniel Sánchez

En el proyecto "Diseño e Implementación de una Fuente de Datos Tipo Snort Basado en la Arquitectura SELFNET" se definieron tres etapas a seguir para la realización del proyecto.

Daniel, en la primera etapa del proyecto, realizó un curso de SDN ofertado por la plataforma de educación virtual Coursera, con el fin de obtener todos los conocimientos necesarios a cerca de esta tecnología. Además, pese a la falta de cursos relacionados con NFV, decidió buscar por su cuenta papers que explicaran con detalle esta tecnología. Gracias al Instituto Europeo de Normas de Telecomunicaciones (del inglés European Telecommunication Standards Institute ETSI) que promueve la estandarización de NFV, pudo encontrar toda la información necesaria. Este trabajo de fin de grado forma parte del proyecto H2020, por lo que el director pudo proveer de toda la información necesaria relacionada con la arquitectura de SELFNET.

También dedicó tiempo al estudio del lenguaje de programación Python, gracias a un curso que impartió la Universidad Complutense de Madrid. Encontró multitud de manuales y ejemplos de uso de las principales estructuras utilizadas en Python, que serían de utilidad para la realización de los scripts propuestos en la fase de desarrollo.

Ya que todos los componentes generados en la fase de desarrollo iban a estar en servidores basados en Ubuntu, fue necesario el estudio tanto de comandos triviales como de funciones para poder elaborar scripts de mayor complejidad.

Una vez finalizada la etapa de recolección de información, y adquirir los conocimientos necesarios para poder llevar a cabo este proyecto, se procedió a

realizar la segunda etapa.

La segunda etapa implicó el diseño y la implementación de una fuente de datos tipo SNORT. Daniel se encargó de realizar el script en Python denominado SNORTDataSource.py. La realización de este script fue gracias a los conocimientos adquiridos en la primera etapa del proyecto, ya que Daniel desconocía por completo el lenguaje de programación Python y las funciones principales del sistema operativo UNIX. También se encargó de realizar la instalación y configuración de la red, para llevar a cabo las pruebas necesarias y obtener los resultados pertinentes.

Por último, la tercera etapa supone la redacción de la memoria, en la que Daniel contribuyó en la redacción de los capítulos de SDN, NFV y SELFNET así como la introducción y los apartados que necesitaron de una traducción al inglés.

8.2 Gerardo Reyes

Para el desarrollo del proyecto "Diseño e Implementación de una Fuente de Datos Tipo Snort Basado en la Arquitectura SELFNET" se participó previamente como oyente en el proyecto “Diseño e Implementación de una Herramienta de Visualización para el Análisis en Tiempo Real de Redes SDN/Openflow”. Este fue el principio de una serie de etapas que se mencionan a continuación.

Gerardo, realizó un curso de SDN ofertado por la plataforma de educación virtual Coursera, con el fin de obtener todos los conocimientos necesarios a cerca de esta tecnología. Además participo en la puesta en marcha de la arquitectura SDN/Openflow sobre una Mininet realizada por el proyecto del año pasado.

Este trabajo de fin de grado y el trabajo del año pasado forman parte del proyecto H2020, por lo que el director pudo proveer a Gerardo de toda la

información necesaria relacionada con la arquitectura de SELFNET.

Con vistas a la continuación del proyecto SELFNET, Gerardo participo en el curso Big Data: Análisis de datos con Python, ofertado por la Universidad Complutense de Madrid de forma presencial y con duración de dos semanas.

En las siguientes etapas se procede a diseñar las estructuras de topología en bus, las cuales pasaron por 3 versiones antes de llegar a la estructura final que se utilizó en este proyecto. Se tuvo que aprender las funcionalidades que Apache Kafka brinda así como la API que ofrece para el lenguaje Python.

Se diseñan scripts en BASH para interactuar con Linux y Python así como la creación de tareas automáticas y la implementación de servicios en Ubuntu para las primeras versiones de la arquitectura de este proyecto. Se corrige los diversos problemas que iban apareciendo en el sistema operativo a medida que se integraban nuevos paquetes de Python 2.7.

Se generan una estructura en Python compuesto de librerías y módulos para no sobrecargar los scripts y sea más fácil la integración de nuevos módulos al proyecto. También se crea un entorno gráfico para poder controlar la ejecución de Kafka y la monitorización en tiempo de ejecución de los datasets que se iban generando además de la monitorización en tiempo real del uso de la CPU.

Se prueba la integración de una base de datos en MySQL aunque para este proyecto no se ha llegado a usar, existe el modulo de conexión mediante Python para sus futuros usos. También se ha implementado la utilización de paquetes gráficos y numéricos en Python y se adaptan a la versión Python 2.7 mediante modificación de algunas librerías de origen y cambiando la codificación del sistema de ASCII a UTF-8.

Para finalizar, en la redacción de la memoria, en la que Gerardo contribuyó en la redacción de los capítulos de fuente de datos snort, pruebas y resultados y conclusiones y trabajos futuros.

RESUMEN EN INGLÉS

9. INTRODUCTION

Due to the increase in data traffic and new communication needs, a new paradigm emerges. Within this context arise technologies such as Software Defined Networks or Software Defined Networking (SDN) and Network Function Virtualization (NFV) that are the basis for future 5G networks. SDN is based on the separation between the data plane and the control plane, obtaining an architecture in three layers:

1. **Data plane** supposes the processing of all the traffic that circulates through the network through switches.
2. **Control plane** is in charge of managing the behavior of the network through the SDN controllers.
3. **Application layer**, where there are applications that allow automating configuration tasks, deploying and providing new services within the network, etc.

The communication between the different layers is carried out through two interfaces: NorthBound and SouthBound API.

1. **NorthBound API**: is responsible for communicating the controller, with high-level programmed applications.
2. **SouthBound API**: is responsible for communicating the network devices that make up the data plane, with the controller that is part of the control plane.

On the other hand, the incorporation of NFV means to stop developing network applications under specific hardware, which makes it impossible to use them in other network environments. The NFV architecture is composed of three main modules: NFV Infrastructure (NFVI), VNF and NFV Manager and Orchestrator

(NFV M & O).

1. **NFVI**: this module contains the hardware and software components that make up the environment in which Virtual Network Functions (VNF) are deployed.

2. **VNF**: is the software implementation of a network function that is able to operate on the NFVI.

3. **NFV M&O**: its task is to orchestrate and manage the life cycle of physical and/or software resources, which support infrastructure virtualization and life cycle management of VNFs.

Both technologies are integrated into the SELFNET project, which designs and implements an autonomous network infrastructure management architecture. And whose architecture is based on six layers:

1. **Infrastructure Layer**: it is based on the designs and components that are currently being used in the 5G networks.

2. **Virtualized Network Layer**: the network functions and virtual machines are instantiated in order to provide functionalities to the users.

3. **SON Control Layer**: includes the elements for data collection from the different sensors and the functions that perform actions on the network.

4. **SON Autonomic Layer**: responsible for providing the automation of network functions.

5. **SON Access Layer**: implements an interface to manage the services exposed in SELFNET.

6. **Management and Orchestration Layer**: manages the control of the network infrastructures available in the architecture.

Finally, the design and implementation of an SNORT data source will be

defined. This is a data source capable of receiving notifications of a SNORT type sensor, after which a data processing task will be performed in order to obtain an output file that will be used to obtain metrics that are used in higher layers of aggregation and correlation.

9.1. Research Objective

In order to guarantee the correct operation of the network, it will be necessary to receive and process the information collected by the network sensors. The purpose of this project is to receive information from these sensors through a data bus. Next, the necessary information will be filtered and data will be added from a configuration file. Finally, the resulting file will be sent by a bus exactly the same as the previous one, so that the upper layers obtain the normalized metrics.

9.2. Related work

There are a variety of applications that integrate technologies such as SDN and NFV. Some examples are described in Table 1.1.

Application	SDN	NFV	Description
DEMon [AP1]	Yes	No	Used to monitor the Microsoft data center.
F5´s Big IP [AP1]	Yes	No	It is an application that monitors different types of threats in the network.
Kemp [AP2]	Yes	No	It provides adaptive load balancing as a function of the traffic on the switches.
Juniper [JUN15]	No	Yes	It combines an application management system in the cloud and a supporting reference architecture.
Masergy [TRU14]	No	Yes	Provides a network behavioral analysis and a correlation-based security platform.
Velocloud [VEL]	No	Yes	Simplifies network management by automating deployment and improving performance.
FastStream Platform [FST]	Yes	Yes	Includes OpenFlow client, hybrid routing, switching framework, and OpenFlow test automation framework.
Mellanox [MEL16]	Yes	Yes	Increase performance through smart adapters, increase scalability and reduce deployment cost.
Brain4Net [B4N16]	Yes	Yes	Provides adaptation services to SDN and NFV.

Table 9.1: Comparison of SDN and NFV applications.

9.3. Document structure

This project is divided into eight chapters, described below.

Chapter 2 explains in detail the operation and architecture of SDN networks.

Chapter 3 defines the concept of NFV and the three main modules of its architecture.

Chapter 4 describes the SELFNET project along with its architecture and its two main tasks are the monitoring and discovery of the metrics generated by the virtualized infrastructure.

Chapter 5 describes the design and implementation of a SNORT data source.

Chapter 6 explains the tests performed and the results obtained after the implementation of a SNORT data source.

Chapter 7 presents the main conclusions and future work. In the same way, a summary is described with the relevant information of the study.

Finally, Chapter 8 shows the division of labor performed by each of the project members.

10. CONCLUSIONS AND FUTURE WORK

10.1 Conclusions

In the present project, an SNORT-type data source has been designed based on the SELFNET architecture. The data source has the ability to collect information from virtual sensors and send them to the collector. At the same time, the metrics are normalized to a generic format capable and the results are sent to the collector.

To achieve this goal, several evaluations had to be carried out until a balance between reliability and performance was found. This has been achieved in large part by the use of Kafka that does the task of messaging, storage and processing of streams, although its strong point is the redundancy of errors thanks to the handling of the position of the consumer (offset), avoiding possible failures when processing the message or connection loss, as in presenting some of these problems Kafka will resend the message again.

Once the problem of error redundancy has been solved, it focuses on improving the performance of data processing and after several evaluations and after having contrasted results, it can be ensured that to achieve a higher yield and productivity in the data transfer is essential to have the necessary hardware resources for faster and more efficient processing.

Therefore, finally it can be concluded that the increase of data analyzed per unit of time, is due to the improvement in the HW of the machine that has processed the messages (in this case it is the SNORTDataSource server). If higher performance is to be expected, it is also necessary to implement more parallel bus servers, since the tests performed in Scenarios 1 and 2 have shown that the performance of the SNORT Data Source has on average 9 jobs processed per second. Also, the CPU and Memory consumption show an average load of 7

and 25 percent of use respectively. Finally, it is concluded that the performance, not improved by more *topics* or consumers is increased, but depends to a large extent on improvements in hardware resources that can be implemented.

10.2 Future Work

In the framework of the design of an information exchange system, there is an infinite number of possibilities for optimization in all aspects of the system. But if you intend to continue research on this system there is still a lot of testing to do, for example, implementing Kafka in a server cluster to improve performance with the use of various topics, test the information balancing option in Kafka and thus avoid possible falls from servers and improve error redundancy by having the same topic on multiple servers.

In addition, thanks to the metrics generated by the project in the evaluation phase, data sent by the sensors that were discarded when sending the configuration files to the collector has been collected, this can be used for future projects to collect datasets or use those generated by this project to make evaluations of the network or use the datasets for an endless number of purposes such as: packet filtering in a conventional network or SDN, decision making when changing the topologies or simply to do Predictive analytics using Big Data.

Anexo A

Ejemplo de estructura para un fichero XML que se recibe de SNORT.

```
<IDMEF-Message xmlns="http://iana.org/idmef" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0" xsi:schemaLocation="http://iana.org/idmef http://webs.um.es/mgilperez/reclamo/schema/IDMEF-v1.0.xsd">
  <Alert messageid="1">
    <Analyzer analyzerid="tcpdump_inside">
      <Node>
        <name>tcpdump_inside</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0xbc6f57c7.0x00000000">2000-03-07T11:26:15Z</CreateTime>
    <Source spoofed="unknown">
      <Node>
        <Address category="ipv4-addr">
          <address>202.77.162.213</address>
        </Address>
      </Node>
      <Service>
        <port>49212</port>
      </Service>
    </Source>
    <Target>
      <Node>
        <Address category="ipv4-addr">
          <address>172.16.115.20</address>
        </Address>
      </Node>
      <Service>
        <name>tcp</name>
        <port>23</port>
      </Service>
    </Target>
    <Classification text="unknown"/>
  </Alert>
</IDMEF-Message>
```

Anexo B

Ejemplo de estructura para un fichero JSON que se envía al colector.


```
{
   "reporterDescription":
    {
      "reporterEpochTime": "1491487846000",
      "reporterMAC": "00:ff:12:34:56:34",
      "reporterAppType": "SMA",
      "reporterHostname": "SnortAgent",
      "reporterIP": "193.136.93.101"
   },
   "reporterID": "reporterHostName=SnortAgent/reporterIP=193.136.93.101/reporterAppType=SMA",
   "resourceDescription":
    {
      "signatureRevision": "1",
      "sensorId": "0",
      "generatorId": "1",
      "signatureId": "10000001",
      "classificationId": "31"
   },
   "resourceType": "snort-dpi-alert",
   "Timestamp": "2000-03-07T11:26:15Z",
   "EventDefinition":
    [
      {
         "name": "srcIP",
         "value": "202.77.162.213"
      },
      {
         "name": "dstIP",
         "value": "172.16.115.20"
      },
      {
         "name": "dstName",
         "value": "tcp"
      },
      {
         "name": "classification",
         "value": "unknown"
      }
   ],
   "dataType": "event",
   "MessageID": "1",
   "resourceID": "snortinstanceX.b-dpi"
}
```

BIBLIOGRAFÍA

[AK] Apache. "Kafka". https://kafka.apache.org/ documentation/ #design.

[AP1] S. Makam. "SDN OpenFlow Commercial Applications - Part 1". https://sreeninet.wordpress.com/2014/08/03/sdn-commercial-applications

[AP2] S. Makam. "SDN OpenFlow Commercial Applications - Part 2". https://sreeninet.wordpress.com/2014/08/09/sdn-openflow-commercial-applications-part-2

[ATK] D. Asturias. "9 Types of Software Defined Network Attacks and How to Protect from Them". http://www.routerfreak.com/9-types-software-defined-network-attacks-protect.

[AZ] Apache. "Zookeeper". https://zookeeper.apache.org/

[B4N16] Intel Corporation. "Brain4Net and Intel Bring Integrated NFV/SDN Technology to Multiprotocol CommSP Networks", 2016.

[CV17] A. L. Valdivieso Caraguay; L. J. García Villalba. "Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks. *Sensors"*, vol. 17, no 4, pp. 731, 31 March 31, 2017.

[ETSI14] European Telecommunications Standards Institute. "Network Functions Virtualization (NFV); Management and Orchestration". ETSI GS NFV-MAN 001 v1.1.1, December, 2014.

[FST] FastStream Technologies. "SDN-NFV". https://www.faststreamtech.com/solutions/sdn-nfv

[GARG14] A. GARG. "Network Function Virtualization". PhD Thesis. Indian Institute of Technology Bombay, 2014.

[GUJ16] J. Garay; J. Unzilla; E. Jacob. “Service Description in the NFV Revolution: Trends, Challenges and a Way Forward”. *IEEE Communication Magazine*, 54, pp. 68–74, 2016.

[JUN15] Juniper Networks. “Network Simplification with Juniper Networks Virtual Chassis Technology”. 2000427-003-EN, September, 2015.

[LIT15] A. D. Little. “Reshaping the future with NFV and SDN: The impact of the new technologies on carriers and their networks”. Bell Labs. Alcatel-Lucent, May, 2015.

[MEL16] Mellanox Technologies. “Optimize your Cloud with the Right Network Adapters”, 2016.

[MIN] Introduction to Mininet. https://github.com/mininet/ mininet/ wiki/Introduction-to-Mininet#what.

[MSG15] R. Mijumbi; J. Serrat; J. L. Gorricho; N. Bouten; F. Turck; R. Boutaba. “Network Function Virtualization: State-of-the-art and Research Challenges”. *IEEE Communication Survey Tutor*. *18*, pp. 236–262, 2015.

[NFV1] European Telecommunications Standards Institute. “Network Functions Virtualisation (NFV). Network Operators Perspectives on Industry Progress”. Proceedings of the SDN and OpenFlow World Congress, Frankfurt-Germany, October 15-17, 2013.

[NFV2] European Telecommunications Standards Institute. “Network Functions Virtualisation (NFV). Network Operators Perspectives on Industry Progress”. Proceedings of the SDN and OpenFlow World Congress, Darmstadt-Germany, October 14-17, 2014.

[NFV3] European Telecommunications Standards Institute. “Network Functions Virtualisation (NFV); Management and Orchestration”. ETSI GS NFV-MAN 001 V1.1.1, December, 2014.

[NFV4] European Telecommunications Standards Institute. “Network Functions Virtualisation. An Introduction, Benefits, Enablers, Challenges & Call for Action”. Proceedings of the SDN and OpenFlow World Congress, Darmstadt-Germany, October 22-24, 2012.

[ONF1] Open Networking Foundation. “Software-Defined Networking (SDN) Definition”. https://www.opennetworking.org/sdn-resources/sdn-definition.

[ONF2] Open Networking Foundation. “OpenFlow Switch Specification”. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf. 19, December, 2014.

[RS13] R. Jain, S. Paul. “Network Virtualization and Software Defined Networking for Cloud Computing: A survey”. IEEE Communications Magazine, 51, pp. 24 – 31, 2013.

[RSG] O. Salman, I. H. Elhajj, A. Kayssi, A. Chehab. “SDN Controllers: A Comparative Study”. Proceedings of the 18th Mediterranean Electrotechnical Conference, (MELECON 2016), Limassol, Cyprus, April 18-20, 2016.

[SEL] Selfnet – Framework for Self-Organized Network Management in Virtualized and Software Defined Networks. https://selfnet-5g.eu/

[SD] Sensor Data. https://www.ll.mit.edu/ideval/data/2000/ LLS_DDOS_1.0/ data_and_labeling/LLS_DDOS_1.0.tar.gz

[SNT] Snort. https://www.snort.org/documents

[NCR15] P. Neves, R. Calé, M. Rui Costa, C. Parada, B. Parreira, J. Alcaraz-Calero, Q. Wang, J. Nightingale, E. Chirivella-Perez, W. Jiang, H. Dieter Schotten, K. Koutsopoulos, A. Gavras and M. João Barros. “The SELFNET Approach for Autonomic Management in an NFV/SDN Networking Paradigm”. December, 2015.

[TRU14] D. A. Trumbull. “Masergy Performance Beyond Expectations. Unified Enterprise Security. A Holistic Approach to Integrated, Behavioral-Based Network Security”, 2014.

[VEL] VeloCloud. “Cloud Delivered SD-WAN”. www.velocloud.com